DTIC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE

1

:UMENTATION PAGE

Form Approved OMB No. 0704-0188

AD-A217 447

1b. RESTRICTIVE MARKINGS: NONE

2a. [illegible]

2b. :...DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

4. PERFORMING ORGANIZATION REPORT NUMBER(S):

5. MONITORING ORGANIZATION REPORT NUMBER(S): AFIT/CI/CIA-89-009

6a. NAME OF PERFORMING ORGANIZATION: AFIT STUDENT AT MISSISSIPPI STATE UNIVERSITY

6b. OFFICE SYMBOL (If applicable):

7a. NAME OF MONITORING ORGANIZATION: AFIT/CIA

6c. ADDRESS (City, State, and ZIP Code):

7b. ADDRESS (City, State, and ZIP Code): Wright-Patterson AFB OH 45433-6583

8a. NAME OF FUNDING/SPONSORING ORGANIZATION:

8b. OFFICE SYMBOL (If applicable):

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:

8c. ADDRESS (City, State, and ZIP Code):

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) (UNCLASSIFIED): Standardization of PROGRAM EAGLE - Numerical Grid Generation System

12. PERSONAL AUTHOR(S): Agusto Martinez

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
THESIS/~~DISSERTATION~~	FROM _____ TO _____	88 December	101

16. SUPPLEMENTARY NOTATION: APPROVED FOR PUBLIC RELEASE IAW AFR 190-1
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

DTIC ELECTE FEB 01 1990 S D

90 02 01 016

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
ERNEST A. HAYGOOD, 1st Lt, USAF	(513) 255-2259	AFIT/CI

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

AFIT/CI "OVERPRINT"

Standardization of Program EAGLE - Numerical Grid Generation System

By

Agusto Martinez

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Aerospace Engineering

Mississippi State, Mississippi

December 1988

Standardization of Program EAGLE - Numerical

Grid Generation System

By

Agusto Martinez

APPROVED

Charles B. Cleveland
Professor and Head of the
Department of Aerospace
Engineering

Walter R. Carnes
Director of Graduate
Instruction, College of
Engineering

Professor of Aerospace
Engineering
(Major Professor)

Walter R. Carnes for
Dean of the College of
Engineering

Kenneth R. Hall
Associate Professor,
Graduate Coordinator,
Department of Aerospace
Engineering

Richard D. Koshel
Dean of the Graduate School

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. Joe F. Thompson for his guidance and direction.

To my wife whom I love.

A. M.

Mississippi State University

December 1988

Abstract

Agusto Martinez, Master of Science, 1988

Major: Aerospace Engineering, Department of Aerospace Engineering

Title of Thesis: Standardization of Program EAGLE - Numerical Grid Generation System

Directed by: Dr. Joe F. Thompson

Page in Thesis: 101 Word in Abstract: 122

Abstract

This thesis describes the conversion of Program EAGLE - Numerical Grid Generation System to ANSI FORTRAN77 standards; The Development of the NAMELIST input emulator for Program EAGLE; and the modifications and additions allowing interactive/non-interactive execution of Program EAGLE. All the work being done on an IRIS 4D/70GT computer graphics workstation. Description of additional user supplied inputs for improved input flexibility is also given as well as a discussion of the logic implemented in the routines used by the NAMELIST input emulator.

Results are presented for three generic airframes demonstrating the capabilities of Program EAGLE as an IRIS 4D/70GT computer graphics workstation. Results discussing the porting of the standardized version of the EAGLE code to a SUN 4/280 computer system are also presented.

Keywords: computational fluid dynamics
Computational aerodynamics

Table of Contents

List of Figures

Chapter 1

Introduction

With the advent of large-scale computing machines (Cray X-MP, CRAY 2, etc.), numerical techniques in computational fluid dynamics (CFD) are now able to solve for the flow field surrounding complex airframe configurations[1]. Numerical grid generation has been cited as a major pacing item for realistic aircraft/missile applications[2] and enables researchers to discretize the domain about arbitrarily-shaped geometries.

Considerable progress has been made in surface (boundary) and grid (mesh) generation[3-6]. These advances allow for the application of multi-block elliptic grid generation techniques to complex aerospace vehicles as well as to other complex configurations from other disciplines (i.e. electromagnetics, hydrodynamics, etc.).

In 1987, the CFD community saw the introduction of a generalized three-dimensional arbitrary geometry grid generation code called Program EAGLE - Numerical Grid Generation System[7,8]. Program EAGLE is a composite (multi-block), algebraic and elliptic grid generation system designed to discretize the domain in or around any arbitrarily shaped three-dimensional region. The code combines a three-dimensional, surface generation scheme with a multi-block, three-dimensional boundary-conforming elliptic grid generation scheme.

The surface generation system of Program EAGLE serves as a front-end to the grid generation system. The surface generation routine develops surfaces (or curves in 2D) to be input to the grid generation routine as segments of the boundary of the region within which the grid is to be constructed[7,9].

The grid generation system is a general two or three-dimensional algebraic and elliptic grid generation routine based on a block structure, which allows any number of blocks to be used to fill an arbitrary two or three-dimensional region[10]. Any block can be linked to any other block (or to itself) with complete (or lesser) continuity across the block interfaces as specified by input. This routine uses an elliptic generation system with automatic evaluation of control functions. These are evaluated either directly from the initial algebraic grid and then smoothed, or interpolated from the boundary point distributions[7,8,9,10].

Other features of the surface generation routine and the grid generation routine are discussed in detail in References 9, 10, 11, and 12.

Program EAGLE was initially developed to execute on large-scale computing machines (supercomputers) in a non-interactive environment[7,8]. Since its introduction, though, interest has increased in developing interactive grid generation programs to execute on engineering computer workstations, as well as mini-super computers and also supercomputers (such as the Cray 2). The increased power and capabilities of these computer systems have further sparked the interest of computational fluid dynamicists and computational aerodynamicists in developing numerical grid generation techniques in an interactive environment.

Thus, the objective of this work is to develop an interactive/non-interactive, portable, user-oriented version of the EAGLE code on a state-of-the-art computer workstation, and to demonstrate the capabilities of the interactive version of the code using complex aerospace configurations. A design criterion was that changes made in the code be transparent to the user and not require any changes of the inputs to the EAGLE code.

The purpose of this thesis is to describe this effort to knowledgeable users of Program EAGLE interested in using the standardized interactive version of the EAGLE code. This thesis can and should be considered as a user's manual of the standardized interactive version of the EAGLE code and as a supplement to the user's manual of Program EAGLE in Reference 9.

Chapter 2

Developments

Program EAGLE - Numerical Grid Generation System was originally developed for the Cray 1 and Cray X-MP supercomputing systems[7]. The most recent version of Program EAGLE has continued to be developed on the Cray X-MP and Cray 2 computer systems[9]. The code is written in the FORTRAN language available on these systems and uses the NAMELIST input extension also available on these systems[7,9].

Although Program EAGLE is written in the FORTRAN language, it does not completely follow the FORTRAN77 standards[13,14]. Following these standards will allow codes to be ported over to other computer systems having the FORTRAN77 language compiler in a much easier fashion, i.e. codes can compile and execute with little or no modifications on different computer systems.

The other problem of Program EAGLE is that it uses the Cray NAMELIST Input extension[15]. This feature is very beneficial in that it allows users to supply the inputs into the code by specifying the appropriate variables with their respective values in any order in the input line. This makes the list of inputs much easier to understand and reduces the reliance on the user's manual that would be required if the inputs were structured (i.e., list - directed and/or formatted inputs). As mentioned earlier, NAMELIST input is an extension and not a standard of FORTRAN77. Therefore, not all computer systems carrying the FORTRAN77 compiler have this feature or extension (for example, computer workstations, such as the IRIS-4D/70GT, and others).

To be able to satisfy the stated objectives, then, requires the EAGLE code to be converted to the FORTRAN77 language standards. This also means either elliminating the NAMELIST input feature or writing a NAMELIST input emulator (in FORTRAN77 standard). To ensure the changes made in Program EAGLE are transparent to the user required that a NAMELIST input emulator be written.

2.1 - Conversion

Converting Program EAGLE to FORTRAN77 standards[13] required that both routines of the numerical grid generation system (i.e., surface and grid generation routines) be converted independently and compiled correctly. Since both routines have similar logic, converting the first one was the most time consuming and most difficult, the reason being that knowledge and experience had to be gained of the logic and the routine's operation. Once this knowledge and experience was gained, the second routine was converted quite easily and quickly to FORTRAN77 standards.

The most obvious first step was to eliminate the Cray NAMELIST input extension within the codes. The next step required changes in using Hollerith (or character) strings. FORTRAN77 standards do not accept Hollerith strings but does use character strings instead. This was the major portion of the conversion.

Converting from Hollerith to character manipulation required all character, integer and real variables to have only character, integer and real values, respectively. This was accomplished by specifying variables of type CHARACTER to be character variables, variables of type

INTEGER to be integer variables and variables of type REAL to be real variables. To allow the user the flexibility of specifying character values for integer and real variables, the character values are converted to unique integer (or real, respectively) values. These values are set in PARAMETER or DATA statements in the main portion of each routine. Note that these unique numbers can and may need to be changed to accommodate the computer system on which these routines are to execute.

FORTRAN77 does not allow character values to be embedded in common blocks where integer and/or real variables exist within the same common block. Therefore, all character variables were removed from these common blocks and passed to all the necessary routines through the CALL and SUBROUTINE statements. Separate common blocks for these character variables could have been included. But, as the development of the NAMELIST input emulator was initiated, it was easier to include the character variables in the CALL and SUBROUTINE statements instead of common blocks.

With the major part of the conversion completed, other smaller modifications were still needed to allow proper compilation and execution. One of these included the removal of various DATA statements for variables residing in common blocks. FORTRAN77 does not allow initialization of variables residing in common blocks by DATA statements unless the BLOCK DATA subprogram feature is used. Again, because of the development of the NAMELIST input emulator, it was easier to initialize these variables using executable statements at the beginning of the program instead of using the BLOCK DATA subprogram feature of FORTRAN77.

Other changes to the routines were made to follow suggestions and guidelines given in Reference 14. One of these is to use the generic names of intrinsic functions wherever possible. Where intrinsic functions are used within the routines, the names of these functions were changed to their generic names from their specific names wherever possible.

Another change included the opening and closing of files within the routines instead of allowing the system to open and close them. This gives the user greater flexibility and control over the files needed.

After all these changes were accomplished, Program EAGLE compiled correctly on the IRIS 4D/70GT computer graphics workstation. The next steps involved the development of the interactive logic and finally the NAMELIST input emulator.

2.2 Interactive Logic

The logic of Program EAGLE is such that converting the code to include interactive processing was relatively simple with minimal changes.

First, a status indicator (ISTAT) was included to indicate the error status - zero (0) for no error (default, of course), a negative one (-1) for a "hard" error and a positive one (1) for a "soft" error. A "hard" error occurs, for example, when the problem size has exceeded preset dimensions, thus terminating execution. A "soft" error occurs when the inputs are incorrect or missing. The routines then prompt the

user to retype the input line correctly. A "soft" error behaves just like a "hard" error in the non-interactive or batch mode (Reference 9 has details on the errors that may occur).

This variable ISTAT (either set to 1 or -1) with a GO TO statement replaces the STOP statements found in the original version after an error has been determined. This allows execution to return to a central location within the routines and determines the next course of action depending on the value of ISTAT.

Lastly, the variable IBATCH determines whether interactive or non-interactive processing is to take place. A value of zero (default) indicates interactive execution, while a value of one (1) indicates non-interactive execution of the code. Note that if IBATCH is set to one (1) and interactive execution is attempted, the program will behave as if it is executing non-interactively. Therefore, any errors (hard or soft) will terminate execution. IBATCH also controls the writing of the prompt to standard output. A value of one (1) will not write the prompt.

2.3 Generalized Parser/NAMELIST Input Emulator

The FORTRAN77 does not have a NAMELIST input feature included as a standard. NAMELIST input exists, only as an extension available on some computer systems. Also, NAMELIST input may not be the same between systems having this feature, thereby, making the original EAGLE code difficult to port over to various computer systems, especially systems which do not have the NAMELIST input feature available. To be able to

port Program EAGLE to different computer systems, and enable users to use their previously developed input lists, a generalized parser was written to emulate the Cray NAMELIST input feature[15].

The inclusion of the emulator into the EAGLE code (i.e. into the surface and grid generation routines) did not change, alter or modify the structure of the code; however, additional arrays were required. The addition of these arrays does not increase the memory requirements significantly, since the additional arrays are single-dimension arrays that are two orders of magnitude smaller than the arrays already present in the code.

The inclusion of the emulator does require additions into the calling routines. Although not as simple as adding the NAMELIST input feature into a code as described in Reference 15, the additions are not complex. These additions are non-executable statements such as PARAMETER, DIMENSION, EQUIVALENCE and DATA statements.

Other additions include the two CALL statements to the parsing routines of course. The first CALL statement calls the subroutine RDSTRG which reads in a line as a character string with a maximum of 132 characters. The character string is then passed into the calling routine to be passed to subroutine PARSER, which is the actual NAMELIST input emulator.

Once control returns to the calling routine, the inputs are passed into the necessary variables by use of EQUIVALENCE statements. The fact that EQUIVALENCE statements were needed to communicate from the parser

to the calling routines required that some of the labeled common blocks be removed or modified due to the restrictions imposed by using common blocks and EQUIVALENCE statements.

All the additions and/or modifications made can be identified in the routines by searching for the line(s) containing the date in which the additions (and/or modifications) were done followed by the characters "<gus AM>". The end of the sections with the additions and/or modifications is signified by a line beginning with a "C" followed by a string of asterisks.

Chapter 3

User Inputs

All inputs are given in the NAMELIST input format of the form

E$INPUT ITEM = 'operation', quantity = value, ...$

The "E" in column 1 indicates to print (or echo) the input line onto the output file. If the "E" is omitted, then column 1 must either be blank or have a "C" (meaning to ignore this line).

The first dollar sign ($) indicates the beginning of the input line, while the second one signifies the end of the input line. The name of the NAMELIST follows the first dollar sign, with a required space after the name. INPUT is the name of the NAMELIST in this example. The names used for the NAMELIST in this version of the EAGLE code are SINPUT - for inputs to the surface generation routine - and GINPUT and GOUTPUT - for inputs to the grid generation routine.

ITEM = 'operation' designates the desired operation, while the values relevant to this operation follow (i.e., quantity = value). In this specification, "quantity" refers to the variable or name of the input quantity, and "value" is its value.

Values for arrays can be given in one of two ways. They are

..., quantity = value, value, value,...

or

..., quantity = N*value,...

where N is the number of values. Also, a value for a particular element of an array can be specified by the use of the notation

..., quantity (M) = value

where M indicates the element of the array.

Quantities (variables) of type CHARACTER can only be given character values (with a maximum of 64 characters). Quantities of type INTEGER and REAL can be given integer and real values, respectively, as well as specific character values allowed by the code (see Reference 9, Vols II and III, for these specific values). Presently, the NAMELIST input emulator is not able to accept exponential notation (i.e., 1.0E-04) as input.

For the purposes of clarity, an input line can consist of any number of (continuation) lines where each line has a maximum of 132 characters (columns) and the first column must be blank. This enables an input line to have as many lines as needed, as long as the first line has the dollar sign in the second column and the last line has the second dollar sign signifying the end of the input line. Also, all inputs must be in capital letters unless specifying a filename. The filename in quotes can be lower or upper case letters. And finally, single quotes must be used for all character input values.

Adding the capability of interactive execution has allowed the user to have greater control over the execution of the code by the addition of three new operations.

The first operation allows the user to save a session. The input line looks like the following example:

```
E$SINPUT ITEM = 'SAVE', FILNAM = 'filename' $
```

where "filename" is the name of a file consisting of no more than 64 characters (includes alpha-numeric characters and symbols). This input line should be the first line of input. This feature only operates in the interactive mode. Each input line following this line will be written to the specified file (i.e., "filename").

The second operation allows the user to read in a previously saved session or a previously developed input file. This input line looks like the following example:

```
E$SINPUT ITEM = 'READ', FILNAM = 'filename'$
```

Once again "filename" is the name of the file to be read in by the code. This input line will cause the inputs to be read in from a file by the name of "filename" instead of reading the inputs from the screen (or standard input).

The third operation available allows the user to terminate execution when desired without necessarily providing all the required inputs for a particular problem. The input line looks like the following:

```
E$SINPUT ITEM = 'STOP'$
```

This will terminate execution and save all the necessary files if this line is not encountered within an input file. In other words, the existence of this line in an input file from which the inputs are being read will terminate reading the inputs from this file, and then will expect the rest of the inputs to come from the screen (or standard input).

Other options added to Program EAGLE give the user greater flexibility in choosing the type of output desired. Within the surface and grid generation routines the user has the option of specifying the

filenames on input, or using the default filenames of the files to be written or read in. This can be done by including the following in the input line:

```
..., FILNAM = 'filename',...
```

where "filename" is the name of the file to be written or read in.

For example, an input line to the surface generation routine may look like this:

```
E$SINPUT ITEM = 'COMBINE', COREIN = 1, -5,
     FILEOUT = 1, FILNAM = 'BLOCK1'$
```

This input line will combine the segments stored in cores 1 thru 5 and write them out on the file named "BLOCK1". The grid generation routine can then read in this file with the following example input line:

```
E$GINPUT ITEM = 'FILE', FILE = 11, FILNAM = 'BLOCK1',...$
```

Another additional feature allows the user to select the output from the grid generation routine in the form needed to be read in by the NASA-AMES graphics program called PLOT3D. To select this feature, the user must include the following in an input line:

```
..., OUTER = 'PLOT3D', ...
```

This option is generally selected when the file on which the grid is to be written is specified. For example:

```
E$GINPUT ITEM = 'STORE', FILE = 72, FILNAM = 'FILE72.fmt',
                OUTER = 'PLOT3D'$
```

Selecting the output feature "PLOT3D" will write formatted records to the FORTRAN file 72, which has the filename "FILE72.fmt" in the form needed by PLOT3D.

OUTER = 'PLOT3D' allows two options. One option allows the user to write to the file without a blanking array (default), and the other option will allow the user to write to the file with a blanking array. To select the second option the user must specify the operation ITEM = 'BLANK' and its associated quantities and values in the output phase of the grid generation routine. For example, an input file to the grid generation routine selecting the PLOT3D option with blanking may look like the following:

```
.
.
.

E$GINPUT ITEM = 'STORE', FILE = 72, FILNAM = 'FILE72.fmt',
                OUTER = 'PLOT3D' $

.
.
.

E$GINPUT ITEM = 'END' $

E$GOUTPUT ITEM = 'BLANK', BLOCK = 1, START = 1,1,1,

END = 65,1,31, VALUE = 2 $

.
.
.

E$GOUTPUT ITEM = 'END' $
```

In this input list, file 72 will have the filename of "FILE72.fmt", and the records will be written formatted in the form needed to be read in by the program PLOT3D. With the operation ITEM = 'BLANK' selected, file 72 will also contain an integer array called IBLANK specifying which points are to be 'blanked'. In this example, in block 1 the

points starting at 1,1,1 to 65,1,31 will have an integer value of 2 (VALUE=2) stored in the IBLANK array. Any integer value besides zero (0) in the IBLANK array will tell the PLOT3D program to draw that point on the screen. Any integer value can be specified in the quantity VALUE. The default though is one (1).

Chapter 4

Generalized Parser/NAMELIST Input Emulator Operation

As mentioned earlier, Program EAGLE was originally written to use the Cray NAMELIST input extension[7,9], which is not available on all computer systems. Therefore, to ensure uniformity of inputs on all computer systems, and to allow input lists previously developed to be used without making changes to these lists, a generalized parser/NAMELIST input emulator was written. This emulator generally follows the basic rules of use of the NAMELIST input extension specified in Reference 15. Several basic guidelines will be stated here, however, for the purpose of clarity.

All upper-case letters must be used except when specifying filenames. The filenames can be given in lower or upper-case letters. Single quotes must be used everywhere.

An input line can consist of any number of lines, where the maximum number of characters per line is 132. An input line begins with a dollar sign ($) in column two (2), and is terminated with a second dollar sign at the end of the input line, regardless of how many lines make up the input line. Each line after the first line of the input line must start on or after column two (2).

Column three (3) of the first line must begin with the name of the input list (the NAMELIST name), and a blank must separate the name with the actual beginning of the inputs. Blanks can be used throughout to make the inputs more legible, but blanks cannot appear within variable names or between single quotes when specifying character values.

Column one (1) must be blank or have either an "E" or a "C". An "E" in column one (1) indicates to write the input line to standard output. A "C" in column one (1) means to ignore this line; however, this line will be written to standard output.

4.1 - Parameters, Variables and Arrays

The operation of the parser/NAMELIST input emulator is basically divided into two major routines. The first, subroutine RDSTRG, reads in a character string while the second, subroutine PARSER, takes this string, checks it and then parses it to the appropriate variables.

In the routine RDSTRG the variable STRING is the character string to be read in, with the actual maximum character length being specified by the variable ISTRNG. The variable NSTRNG provides the maximum allowed character length for the variable STRING. Note that NSTRNG should be less than or equal to ISTRNG. The variable IREAD determines whether to read from a file (IREAD = 1) or from standard input (default). The variable ISTAT specifies the error status - zero (0) means no error while any other number (say, 1 and -1) indicates an error has occurred. The variable LCT is the actual length of the character string read in. The variables STRING, LCT and ISTAT are passed back to the calling routine.

The character string STRING is then passed from the calling routine into subroutine PARSER through the CALL and SUBROUTINE statements for parsing. The parameters NVAR, NCHAR, NINTG and NREALS are used to dimension the arrays in subroutine PARSER. NVAR is the total number of

variables the user can supply as input. This parameter is the sum of the other three parameters, NCHAR, NINTG and NREALS. These three are the total number of characters, integers and real variables, respectively, that the user can supply as input.

The three parameters NCVAL, NIVAL and NRVAL are the total number of characters, integer and real values, respectively, that can be supplied during input. The difference between these three parameters and the previous set of parameters is that these three parameters contain the sum total of all the elements of the arrays of the input variables. NCVAL, NIVAL and NRVAL can be the same as NCHAR, NINTG and NREALS if all the variables which can be supplied during input contain only one element (i.e., all the variables are not arrays).

The last three parameters are NDAT, NIDAT, and NRDAT. The sum of NIDAT and NRDAT gives NDAT - the total number of integer (NIDAT) and real (NRDAT) data constants set in the calling routine. These are used for the purpose of specifying character values for integer and real variables. The parser then knows to convert supplied character strings into unique integer and real numbers.

The arrays needed by subroutine PARSER are NAMLIS, NDIM1, NDIM2, ICHR, INTGR, REALS, CDATA, IDATA, RDAT, and CPOS, IPOS, RPOS. The character array NAMLIS is dimensioned to NVAR and contains the list of input variable names organized with input variables of type CHARACTER first, type INTEGER next, and type REAL last. This array is used to check the input and see that the supplied variable exists in the NAMELIST.

The integer arrays NDIM1 and NDIM2 are dimensioned to NVAR and hold the lower and upper bound dimensions, respectively, of all the variables. The values within these arrays are organized in the same manner discussed above. A two-dimensional array will have the upper bound of its dimension set to the total number of elements within that array. These arrays are used to ensure that no more than the required number of values are passed into their respective variables.

The arrays ICHR, INTGR and REALS are character, integer and real arrays, respectively, which hold the values of all the variables. These arrays are then passed back into the calling routine and equivalenced to the appropriate variables. These arrays are dimensioned to NCHAR, NINTG, and NREALS, respectively.

The character array CDATA contains the names of the constants set in the calling routine. This array is used to check the input and see that the supplied character inputs to integer and real variables are present within this array. The array CDATA is dimensioned to NDAT.

The arrays IDAT and RDAT are integer and real arrays, dimensioned to NIDAT and NRDAT, respectively. These arrays contain the unique integer and real constants which will be passed to the appropriate variables when character values are supplied to integer and real variables.

The last three arrays are CPOS, IPOS, and RPOS. These three are integer arrays dimensioned to NCHAR, NINTG and NREALS, respectively. The purpose of these three arrays is to hold actual array positions within the three arrays ICHR, INTGR and REALS, respectively. In other words CPOS, IPOS and RPOS act as pointers for the parser to place the values in the proper position in the respective arrays.

The last set of arguments needed by subroutine PARSER consists of two character strings - STRING and NAMLST - and several integer variables. The first character string called STRING is passed from the routine RDSTRG to the calling program and then to subroutine PARSER. The second character string, NAMLST, holds the name of the NAMELIST input. This variable can be an array holding different names of the NAMELIST input. This is demonstrated in the grid generation routine of the EAGLE code.

The integer variable NDS, indicates where the second dollar sign is located in the character string STRING. If none is found then NDS is set to zero (0), the default.

The variable L maintains a sum total of the lines read in per input line. The default value for L is one (1).

The integer variable INL stores the number of the last input variable found in STRING. INL also has a default value of one (1).

The next variable, ITEXT, keeps track of the number of values supplied for the last input variable in STRING.

The variable IECHO is a print (or echo) indicator which is set to one (1) when an "E" is present in column 1 of the first line of the input line. The default value for IECHO is zero (0).

The variable LS and LT store the character lengths of the character strings STRING and TEXT, respectively. (TEXT stores the name of the input variable.) Both have default values of one (1).

The variable NL serves as a search pointer indicating where in the character string STRING the search last ended. Its default value is one (1).

The next variable NJ stores the total number of elements within an array. This variable is used to ensure that the value stored in ITEXT is less than or equal to NJ. Again, the default value for NJ is one (1).

IEQUAL indicates whether an equal sign is present. A value of one (1) means an equal sign was found, while a value of zero (0) means no equal sign is present.

The penultimate variable, NUM, indicates to multiply the given value by an integer number when the value for NUM is one (1).

The last variable in the argument list (ISTAT) is the status indicator discussed earlier.

All these parameters, arrays and variables are needed by the generalized parser/NAMELIST input emulator to successfully parse the inputs. Only the character string STRING and the three arrays storing all the inputs - ICHR, INTGR and REALS arrays - are used by the calling routine.

4.2 Program Configuration/Set-Up

To be able to use this emulator, several non-executable statements must be added to the calling routine, as well as two CALL statements to the routines which read in the lines of input (subroutine RDSTRG) and which parse these inputs to the appropriate variables (subroutine PARSER). Appendix A has a sample listing of all the statements which need to be added to the calling routine. The following paragraphs will discuss the needed statements.

To begin with, several arrays and the variable STRING must be specified as type CHARACTER. The character length of STRING must match the value specified for the variable ISTRING in the PARAMETER statement. The value for NSTRING can be set to ISTRNG or to a lesser value as evidenced in Appendix A.

Next, PARAMETER statements containing the values for the variables NCHAR, NINTG, NREALS, and NCVAL, NIVAL, NRVAL and NRDAT are required. Also, PARAMETER statements will be needed to initialize the variables NVAR and NDAT. NVAR is defined by the equation

$$NVAR = NCHAR + NINTG + NREALS$$

and NDAT is defined by the equation

$$NDAT = NIDAT + NRDAT$$

These statements set up the dimensions needed by the arrays used by the routine PARSER.

The next item is to dimension the arrays needed by the parser. The arrays NAMLIS, NDIM1 and NDIM2 are dimensioned to NVAR. The arrays ICHR, INTGR and REALS are dimensioned to NCVAL, NIVAL and NRVAL, respectively. The arrays CPOS, IPOS and RPOS are dimensioned to NCHAR, NINTG and NREALS, respectively. The last three arrays CDATA, IDAT and RDAT are dimensioned to NDAT, NIDAT and NRDAT, respectively.

After the DIMENSION statements comes the setting up of the communication between the three arrays returned by the parser - ICHR, INTGR and REALS - and the variables used by the calling routine. This is accomplished by using EQUIVALENCE statements. One set of these statements passes the values in the character array ICHR into the appropriate charac-

ter variables. The next set passes the integer values in the array INTGR into the appropriate integer variables. And finally, the third set of statements passes the real values in the array REALS into the appropriate real variables.

The last set of EQUIVALENCE statements passes unique integer and real numbers to the parser through the arrays IDAT and RDAT, respectively. This last set of statements is required only if character values are to be supplied during input to integer and real variables. These unique numbers should be set either in PARAMETER statements or DATA statements.

The last set of statements - DATA statements - initialize the arrays NAMLIS, NDIM1, NDIM2, CPOS, IPOS and RPOS, as well as initializing the unique numbers and the default values for the variables needed in subroutine PARSER.

After completing the set-up of the non-executable statements, the section which reads the inputs and then parses the inputs must be added. The first CALL statement is to the routine RDSTRG which reads the input (as a character string), checks its length and returns the character string STRING. The second CALL statement is to the routine PARSER where the character string STRING is passed into it, and the input values are returned in the three arrays ICHR, INTGR and REALS. The character string STRING can be used by the calling routine since it is not changed in any way. These CALL statements should be placed after initializing the defaults and prior to testing for correct input values.

4.3 Subroutines and Functions

Subroutine PARSER not only governs the parsing of the inputs but also has the logic necessary to emulate the Cray NAMELIST input feature[15]. This logic is the first thing which the character string STRING goes through.

Initially the search pointer NL is set by using the name of the NAMELIST (NAMLST), and the character length of STRING is also determined and placed in the variable LC.

The first section of the logic ensures that column one (1) is blank or has either an "E" (writes the character string STRING to standard output) or a "C" (the line is ignored and written out to standard output and execution is returned to the calling routine). After this test, a dollar sign ($) is searched for in column two (2). If found then the search continues for the second dollar sign indicating the end of the input line. If this second dollar sign is not found in the first line (L=1) of the input line, the variable LS is set to the character length of STRING (LC). Other subsequent lines will be tested until the second dollar sign is found, thus terminating the input line.

The next test in the logic looks for the name of the NAMELIST (NAMLST) in STRING starting from column three (3) up to the column specified by the search pointer NL. After satisfying this condition, a blank in column NL+1 is searched for, and then NL is reset to its proper position in STRING.

The next part in subroutine PARSER will put together a string of characters in the character string TEXT and then check it against the list (NAMLIS) to determine if the supplied input variable is a valid input. If so, the variable INL specifies which input variable, and sets the maximum number of elements (NJ) and the element position within one of the three arrays (ICHR, INTGR or REALS) for that particular input variable. Finally STRING, LS, NJ and one of the three arrays are passed into one of the three routines (FNDCHR, FNDINT or FNDRL) for parsing, depending on the value in INL.

4.3.1 FNDCHR

Subroutine RNDCHR searches for a character value within the character string STRING, starting from the last position of the pointer (L) to the end of the string (LS) or until the character value has been found, whichever comes first.

The routine begins by searching for the equal sign after the input variable. Generally, the variable L will point to a position in STRING which is a blank or has an equal sign (=) initially. If the equal sign is not found prior to reaching the end of the string, execution will return to the routine calling the parser.

After finding the equal sign, the character value is found in STRING by searching for a set of single quotes ('). Once the first single quote is found, routine CHRVAL is called to get the position of the second single quote within STRING, thus indicating a character value is present. The character value is then stored in the array IC, providing

that the number of values found (ITEXT) does not exceed the number allowed (NJ). The search pointer (L) is then updated, and control is returned to the beginning of the routine to determine if more values are present.

If, after finding the equal sign, a number followed by an asterisk (indicating multiplication) is found, then the upcoming character value will be repeated in the array IC the number of times specified by this number.

This number must be an integer and is stored as a character string in TEXT to be later converted to an integer number. Then the search continues for the asterisk (*). Finding the asterisk, routine CONVER is called to convert the character string in TEXT to a number which is stored as an integer number in ITEXT. If ITEXT is less than or equal to the value in NJ then the variable NUM is set to one (1), and the search continues for the character value as described above. This time, though, the character value is stored ITEXT times in the array IC.

The search is terminated and control is returned to the parser once a comma and a character after the comma is found not satisfying any of the conditions, and the value in ITEXT is less than or equal to the value in NJ.

4.3.2 FNDINT, FNDRL

Subroutines FNDINT and FNDRL search for an integer or real value, respectively, within the character string STRING, starting from the last position (L) to the end of the string (LS), or until all the integer or

real values have been found for the specified input variable. Both routines function similarly, with the exception that subroutine FNDRL requires searching to include a decimal point (.). Also, both routines use the same logic used in subroutine FNDCHR to determine when the input values are being specified. In other words, an equal sign is found first, and then the input values come after the equal sign.

After finding the equal sign, an integer (or real) value is found in STRING by searching for a character between the number zero (0) and nine (9), or a plus (+) or minus (-) sign. In subroutine FNDRL the search also includes a period (.). Once any of these symbols are detected, they are placed in the character string TEXT as characters until a comma (,) is found. Then TEXT is converted to an integer (or real) value by the routine CONVER and stored in the array IC if the number of input values (ITEXT) does not exceed the maximum specified by the variable NJ.

Another piece of logic was inserted to allow input of character values to integer and real variables. The character values are found by searching for a single quote, calling subroutine CHRVAL to determine the position of the second single quote, and then converting this character value to a unique integer (or real) value using subroutine CCI (or CCR), respectively. The unique integer and real values must be preset in the routine calling subroutine PARSER, as well as setting up the arrays needed by the two routines, CCI and CCR (see Section 4.2). After conversion to unique integer (or real) values, they are then stored in the array IC as described earlier.

As discussed in Section 4.3.1, if an asterisk is present after a number has been specified, then the upcoming input value will be repeated in the array IC the number of times specified by the given number.

The search is terminated and control is returned to the parser as described in Section 4.3.1.

4.3.3 CHRVAL

Subroutine CHRVAL finds a character value in the character string STRING by returning the position of the second single quote in STRING.

The first single quote is found prior to calling this routine, and the position is passed into CHRVAL through the variable I. The second single quote is searched for in STRING, and its position is stored in the variable IQ2. If no second quote is found, IQ2 will have a value of zero (0). The character value is then checked for embedded blanks between the single quotes, and, finally, the position of the second single quote is returned to the calling routine in the variable IQ2.

4.3.4 CONVER

Subroutine CONVER converts a character representation of an integer or real number to an actual real number (excluding exponential representation of a real number).

The character representation comes from the calling routine in the character string TEXT. TEXT is first checked to determine if a minus (-) or a plus (+) sign is present. If a minus sign is found then the variable SIG is given the value of -1, while if a plus sign is found SIG will have a value of 1. The beginning (NSTR) of the string TEXT is then set to one plus the present point. If a decimal point is found, the position of the decimal point is returned in the variable NDEC.

Each character in TEXT is converted to a real number by subtracting the assigned value of the ANSI representation of zero (0) from the assigned value of the ANSI representation of each character in TEXT. All the numbers are then added to obtain the value stored in the variable VAR. The result in VAR is a real number which can always be converted to an integer when necessary in the calling routine.

4.3.5 CCI, CCR

Subroutines CCI and CCR convert character values passed in the character string TEXT into unique integer and real values, respectively. These unique values must have been specified in the routine calling subroutine PARSER and passed through the arrays IDAT and RDAT. The array CDATA contains the list of character values to be specified during input for integer and real variables.

TEXT is first checked against the list in CDATA to ensure that the character value within TEXT is allowed. Then the position of the character value within the array CDATA is checked to ensure that the correct unique number is stored in the variable IVAR (VAR in subroutine CCR). IVAR is then passed to the calling routine.

4.3.6 ELEMNT

Subroutine ELEMNT converts a character representation of a number within a set of parenthesis to an integer number indicating the position in the array specified during input.

The character string TEXT contains the input variable (along with the parentheses and number) in character representation. The positions of the left and right parentheses (NL and NR, respectively) within TEXT are determined. Then, between the parentheses, the character representation for the number is stored in the character string NUM. NUM is then passed into the routine CONVER for conversion to an integer value indicating which element in the array. The value is then returned in the variable IVAR.

4.3.7 ILEN, ISIZ

Function ILEN determines the actual number of characters in the character string TEXT.

Function ISIZ counts between the two numbers N1, N2. The sum total is returned in ISIZ.

Chapter 5

Results

5.1 Applications

Three generic configurations were used to demonstrate Program EAGLE's capabilities on an IRIS 40/70GT computer graphics workstation. The first of the configurations is an elliptic cross-section missile[7,16,17]. The second is an ogive-cylinder-ogive with fins[7,18]. And the third is an elliptic airframe with horizontal and vertical control surfaces[8,12].

Each configuration was generated interactively with no other users on the system. Also, each configuration generated used the option in the EAGLE code to execute as if it were executing on a Cray X-MP computer system with solid state desk device (SSD). In other words, files which are assigned to SSD on a Cray X-MP with SSD are now used as regular files. Therefore, CPU times stated may consist of up to 80% I/O time and not actual computation. This statement is made based on similar runs using SSD on the Cray X-MP computer system[19].

5.1.1 Elliptic Missile

This configuration has an elliptic - 3:1 - cross-section throughout the body described by a parabolic curve from the nose to the aft end of

the body. Extending from the end is an attached sting. Figure 1 shows the body of this configuration.

A C-type system is used to discretiize the domain on and around the body. This field grid is described by two (2) blocks (or grids), each block containing one-half of the configuration. Each block has the dimensions of 65x18x31 points - 65 points in the axial direction, 18 points in the radial direction and 31 points in the circumferential direction - for a total of 72540 points (36270 points per block).

Figures 2a through 3b display side and front views of the algebraic grid. Figures 4a and 4b show a perspective view of the entire grid and a closeup view of the grid near the body. The surface and field grids shown in these figures are the same as those presented in Reference 7. This indicates the grids generated on an IRIS 4D/70GT computer system or similar systems are consistent with previously documented work.

The generation of the algebraic grid took 863.81 CPU sec (30 min wall clock time). This breaks down to 1.2×10^{-2} CPU sec/point. An elliptic grid with five iterations was also generated to obtain an idea of the time it takes to generate an elliptic grid. The generation of the elliptic grid took more than twice as long as the generation of the algebraic grid. The times are 2225.01 CPU secs or(68 minutes wall clock), which comes to 3.75×10^{-3} CPU sec/iteration/point. No elliptic grids are shown since they demonstrate little or no difference from the algebraic grids shown in Figures 2a through 4b.

5.1.2 Ogive - Cylinder - Ogive with Fins

This configuration consists of an ogival nose, a cylindrical mid-section and an ogival boattail with fins. Once again, a sting extends from the aft end of the body. Figure 5 displays this geometry.

This configuration also uses a C-type system for the field grid around the body. It consist of four (4) blocks. Each block has a total of 33600 (140x24x10) points, for a sum total of 134400 points for the entire field grid. Figures 6a through 7b display the side and front views of the algebraic grid. Figure 8 shows a perspective view of the grid of one of the blocks near the body. Once again, the grids generated on the IRIS 4D/70GT are comparable to those generated on Cray computer systems as documented in References 7 and 18.

The generation of the algebraic and elliptic grids (5 iterations) took 2022.25 CPU secs (1.5×10^{-2} CPU sec/pt) and 4510.98 CPU sec (3.70×10^{-3} CPU sec/iteration/pt), respectively. The wall clock time for the generation of the algebraic grid was 75 minutes. No wall clock time was observed for the generation of the elliptic grid, although, it can be estimated at approximately 180 minutes. Notice that the times specified in CPU seconds per point for the algebraic grid and CPU seconds per iteration per point for the elliptic grid for the first configuration as compared to this configuration are similar although not the same. The difference can be attributed to the larger number of blocks increasing the I/O time.

5.1.3 Elliptic Airframe with Control Surfaces

This last configuration is of an advanced airframe consisting of an elliptical cross-section for the upper surface of the body, a flat bottom surface, and vertical and horizontal control surfaces. Only half of this airframe was gridded, as seen in Figure 9.

The field grid surrounding this airframe is a H-type system consisting of five (5) blocks, for a total of 111040 points for only half of the airframe and the surrounding domain. The dimensions for each block are as follows:

Block 1 - 70x24x10 (16800) points

Block 2 - 70x24x22 (36900) points

Block 3 - 70x16x31 (34720) points

Block 4 - 20x24x31 (14880) points

Block 5 - 20x16x24 (7680) points

Block 1 encompasses part of the upper surface of the body up to and including one side of the vertical tail. Block 2 has the rest of the upper surface of the body from the other side of the vertical tail to the upper surfaces of the horizontal control surfaces. Block 3 contains all of the bottom surface, including the flat sides of the horizontal control surfaces. The last two blocks - blocks 4 and 5 - encompass the entire domain in front of the airframe. Figures 10a through 13b show front, side and top views of this configuration, and Figures 13a and 13b

show a perspective view of the entire domain and a closeup view of the grid near the body. These grids are also comparable to the grids documented in References 8 and 12.

The algebraic grid took 3038.7 CPU secs to generate, while the elliptic grid (5 iterations) took 6500 CPU secs. With respect to CPU time, the algebraic grid took 2.74×10^{-2} CPU sec/pt and the elliptic grid took 3.8×10^{-3} CPU secs/iteration/pt.

5.2 Porting

After verifying that Program EAGLE worked correctly on the IRIS 4D/70GT computer graphics system, the code was placed on a SUN 4/280 computer system to determine if the code is portable across different computer systems.

After placement of the code on the SUN 4/280 computer system, the FORTRAN77 version of the EAGLE code compiled without errors and without having to make any modifications or changes to the code. This version of Program EAGLE executed correctly for all three configurations described earlier. The CPU times obtained to generate these three configurations on the SUN 4/280 computer system are comparable to those times obtained on the IRIS 4D/70GT.

Chapter 6

Conclusions

The results of this effort show that the FORTRAN77 version of the EAGLE code with its own NAMELIST input emulator works as well as the version written for the Cray systems. The three configurations used to test the FORTRAN77 version of the EAGLE code have demonstrated the capabilities of the code, as well as obtaining results consistent with previous work documented in References 7 through 9, 12, and 18. This version of the EAGLE code can then be used on other systems having the FORTRAN77 compiler with little or no modifications to the code as demonstrated on the SUN 4/280 computer system. As power and speed of computer systems increase, the more plausible it becomes to execute Program EAGLE in an interactive environment, especially when tied to a graphics program allowing the user to "see" the work being achieved.

As with any other effort, this effort is not the one to end all efforts. But it is a beginning of a powerful tool for the computational fluid dynamics community. Further efforts are needed to give the user greater control and flexibility of the execution of the code and allow visualization by the user of the problem at hand.

References

1. Priolo, F. J., Wardlaw, A. B., Baltakis, F. P., and Solomon, J. M., "Inviscid Multiple Zone Strategy Applied to Complicated Supersonic Tactical Missile Configurations," AIAA-85-1813, Aug 1985.

2. Kutler, P., "A Perspective of Theoretical and Applied Computational Fluid Dynamics," AIAA-83-0037, Jan 1983.

3. Mounts, J. S., Martinez, A., and Thompson, J. F., "An Analysis of Elliptic Grid Generation Techniques Using An Implicit Euler Solver," AIAA-86-1766, Jun 1986.

4. Thompson, J. F., "A Survey of Grid Generation Techniques in Computational Fluid Dynamics," AIAA-83-0447, Jan 1983.

5. Thompson, J. F., "A Survey of Composite Grid Generation for General Three-Dimensional Regions," Numerical Methods for Engine Airframe Integration, S.N.B. Murthy and G. C. Paynton (ed.), AIAA, 1986.

6. Thompson, J. F., Warsi, Z.U.A., and Mastin, C. W., Numerical Grid Generation: Foundations and Applications, North-Holland, 1985.

7. Martinez, A., Mounts, J. S., and Thompson, J. F., "Program EAGLE - Numerical Grid Generation System User's Manual," AFATL-TR-87-15, Mar 1987.

8. Martinez, A., Chae, Y. S., and Thompson, J. F., "Program EAGLE - Numerical Grid Generation as Applied to Advanced Airframe Configurations," AIAA-87-2294, Aug 1987.

9. Lijewski, L. E., Cipolla, J., Thompson, J. F., and Gatlin, B., "Program EAGLE - Users Manual", AFATL-TR-88-117; Vols. I, II, III, Oct 1988.

10. Thompson, J. F., "Composite Grid Generation Code for General 3-D Regions - the EAGLE code", AIAA Journal, Vol. 26, No. 3, pg 271, March 1988.

11. Chae, Yeon Seok, "The Construction of Composite Grids for General Three-Dimensional Regions", Ph.D. Dissertation, Mississippi State University, Aug 1987.

12. Jones, Gerald A., "Surface Grid Generation for Composite Block Grids," Ph.D. Dissertation, Mississippi State University, May 1988.

13. American National Standards Institute, American National Standard FORTRAN, X3.9 - 1978 (FORTRAN77). This standard is available from the American National Standards Institute, 1430 Broadway, New York, N.Y. 10018, USA.

14. Balfour, D., and Marwick, D. H., Programming in Standard FORTRAN77, North-Holland, 1985.

15. Cray Research Inc., CFT77 Reference Manual, Publication number SR-0018, Cray Research Inc., 2520 Pilot Knob Rd, Suite 310, Mendota Heights, MN 55120, Sept 1986.

16. Sherera, D. E., Amidon, P. F., Dahlem, V., and Brown-Edwards, E., "Pressure Test of Three Elliptical Missile Body Configurations at Mach Numbers 1.5 to 5.0", AFWAL-TM-84-236-FIMG, Dec 1984.

17. Remotigue, M. G., and Mounts, J. S., "A Semi-Empirical Examination of the Aerodynamics of Elliptic Missile Configurations Using Missile DATCOM", AFATL-TR-85-100, Apr 1986.

18. Cottrel, C. J., and Lijewski, L. E., "A Study of Finned Multibody Aerodynamic Interference at Transonic Mach Numbers", AIAA-87-2480, Aug 1987.

19. Thompson, J. F., Private Communication, Mississippi State University, MS, 1988.

FIGURES

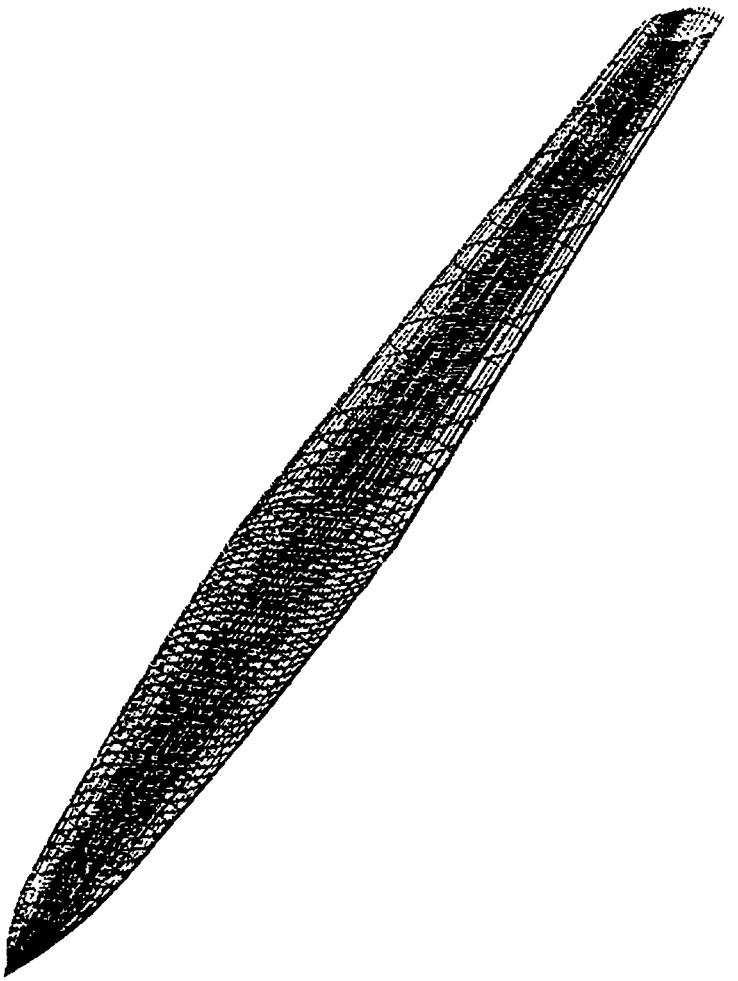

Figure 1 - Elliptic missile surface grid

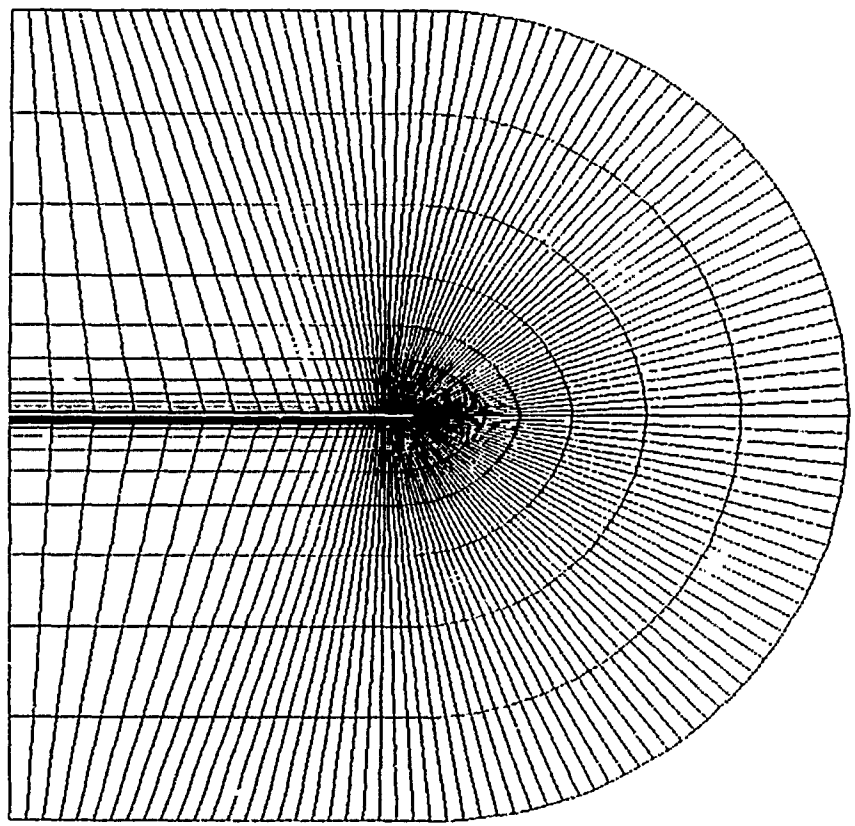

Figure 2a - Side view of elliptic missile field grid

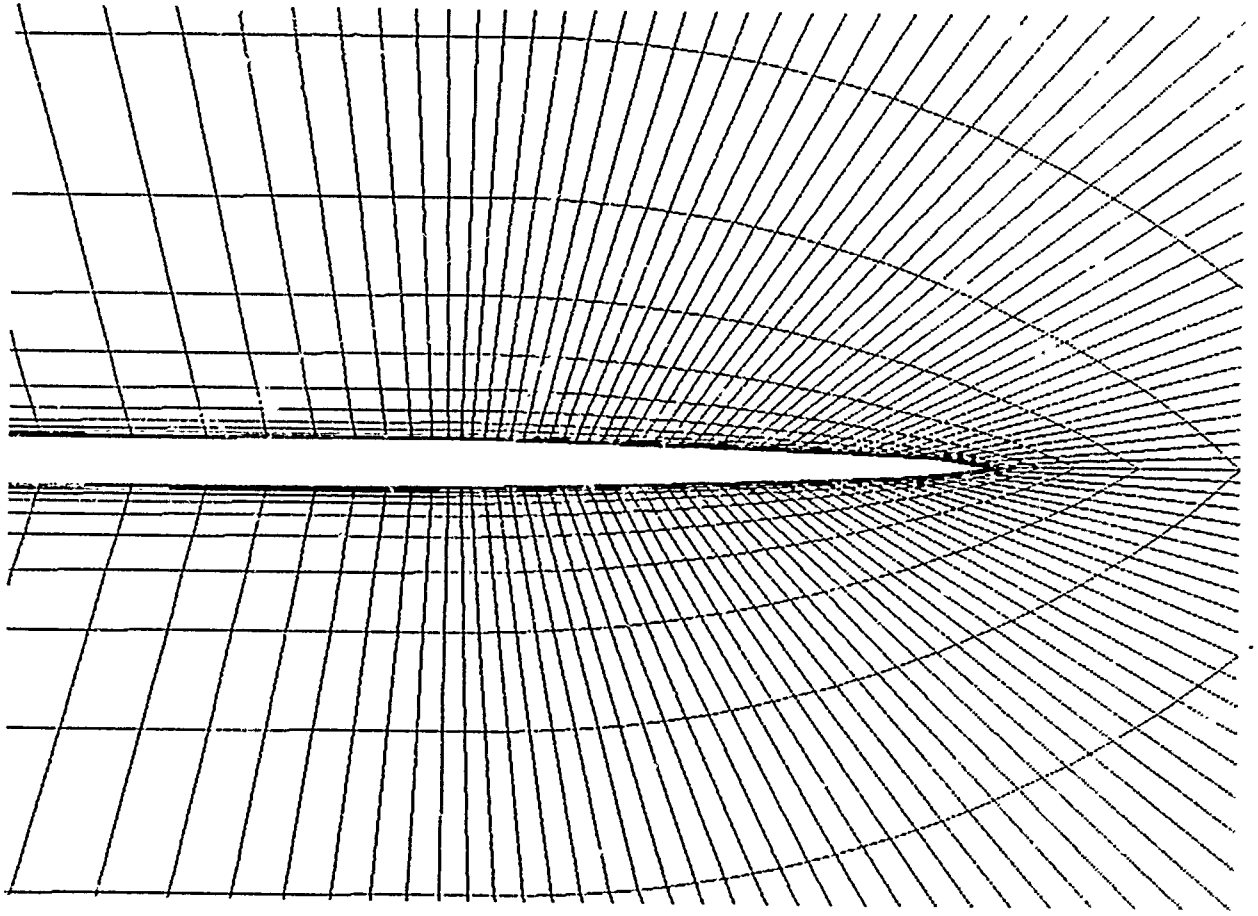

Figure 2b - Close-up side view of elliptic missile field grid

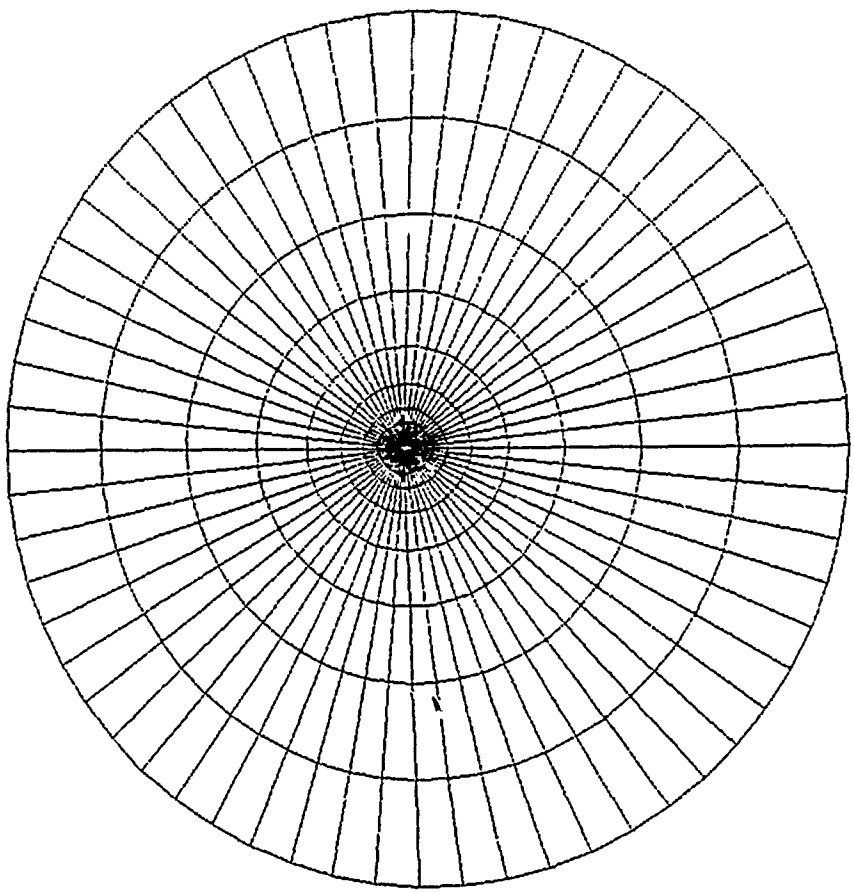

Figure 3a - Front view of elliptic missile field grid

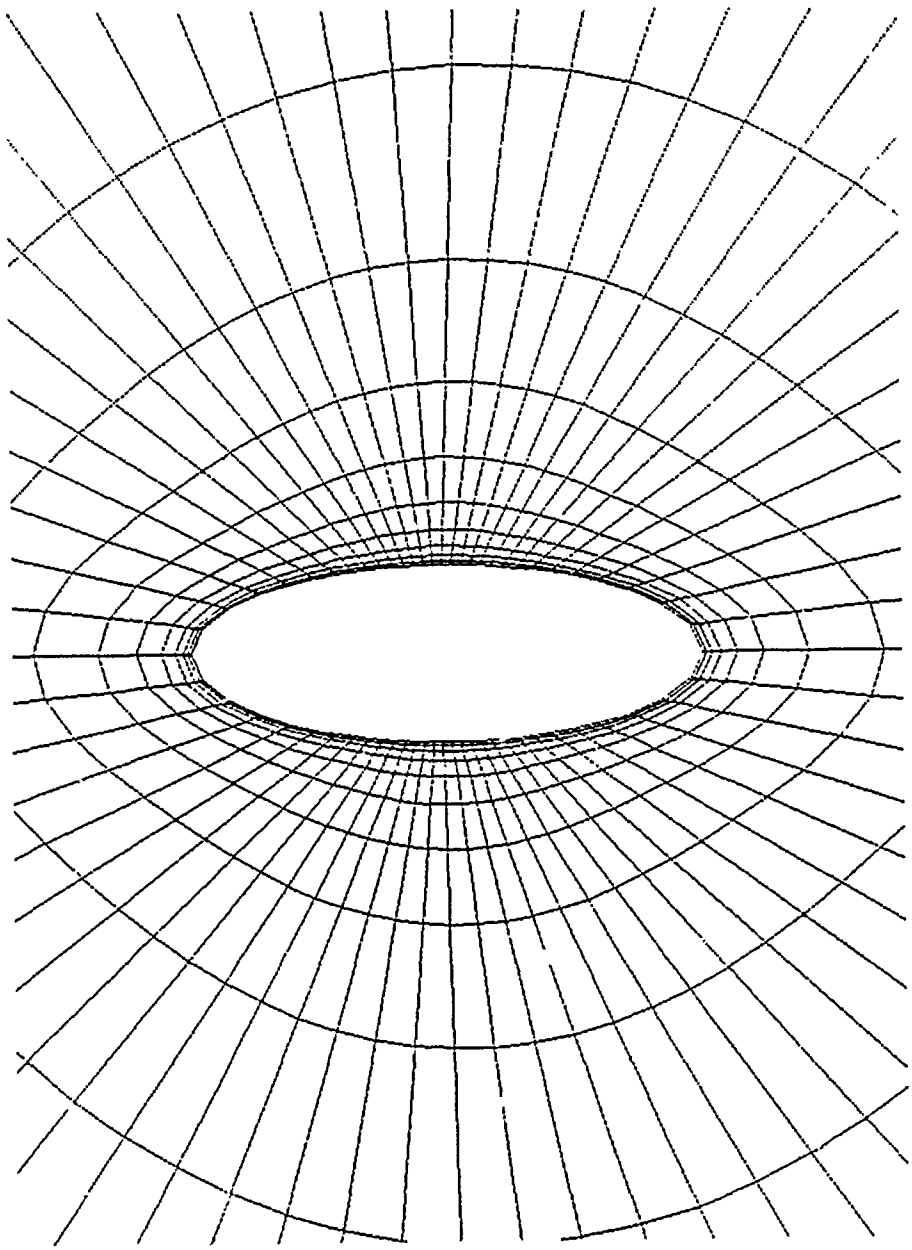

Figure 3b - Close-up front view of elliptic missile field grid

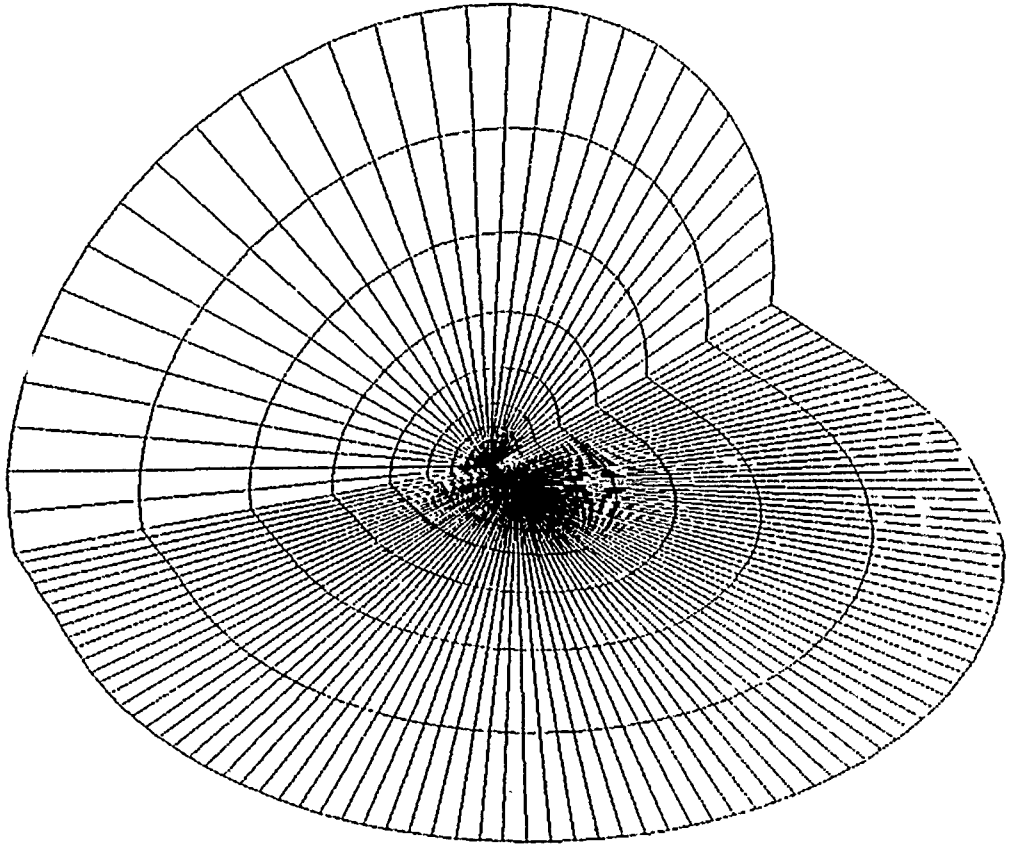

Figure 4a - Perspective view of overall elliptic missile field grid

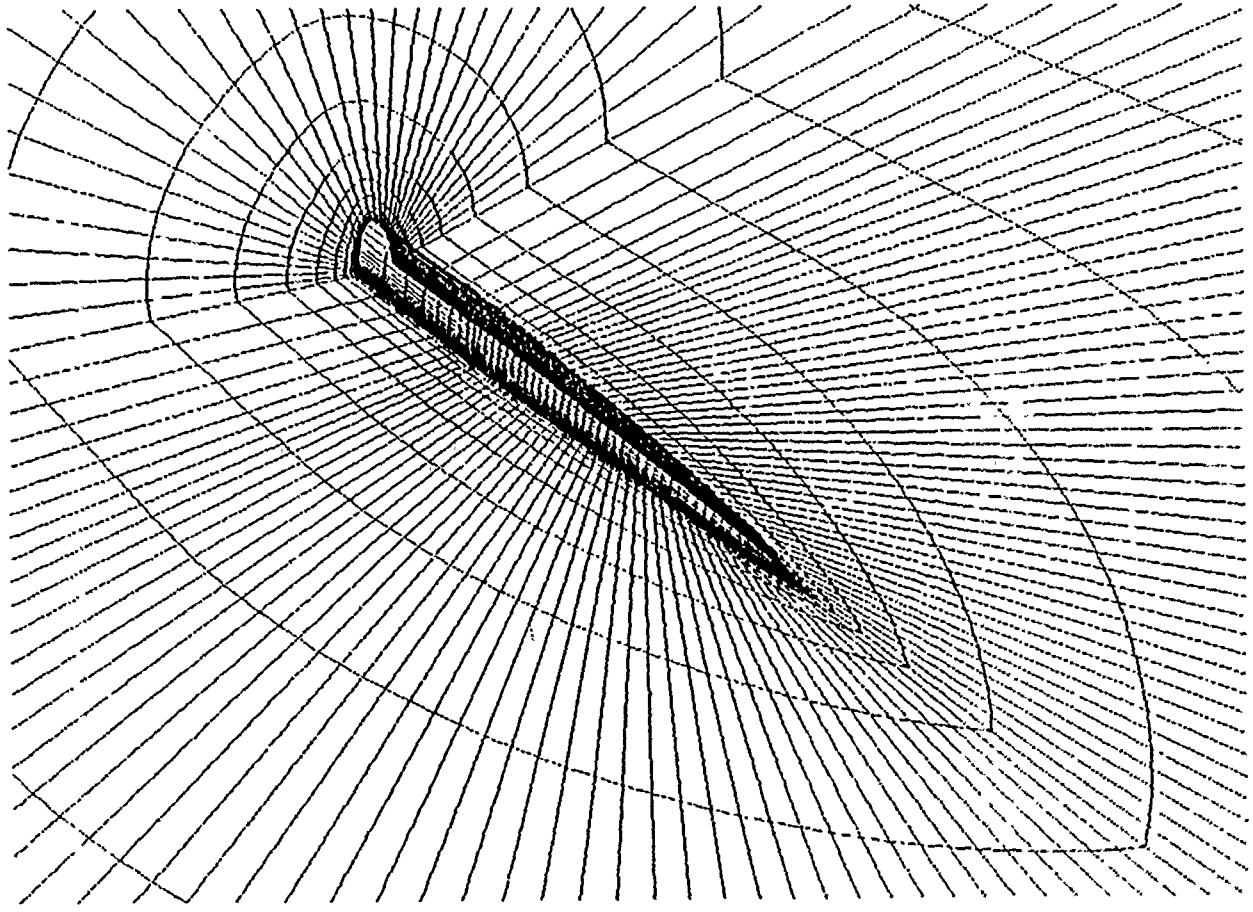

Figure 4b - Close-up perspective view of elliptic missile field grid

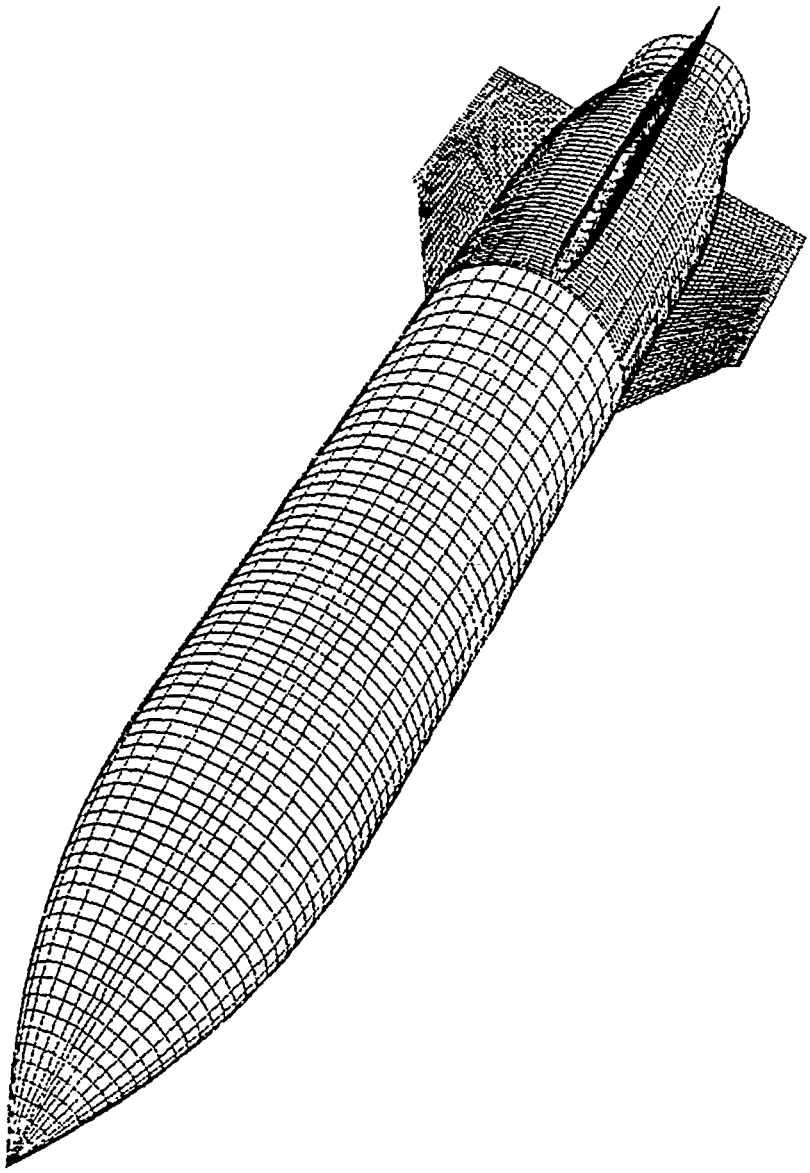

Figure 5 - Ogive-Cylinder-Ogive with fins surface grid

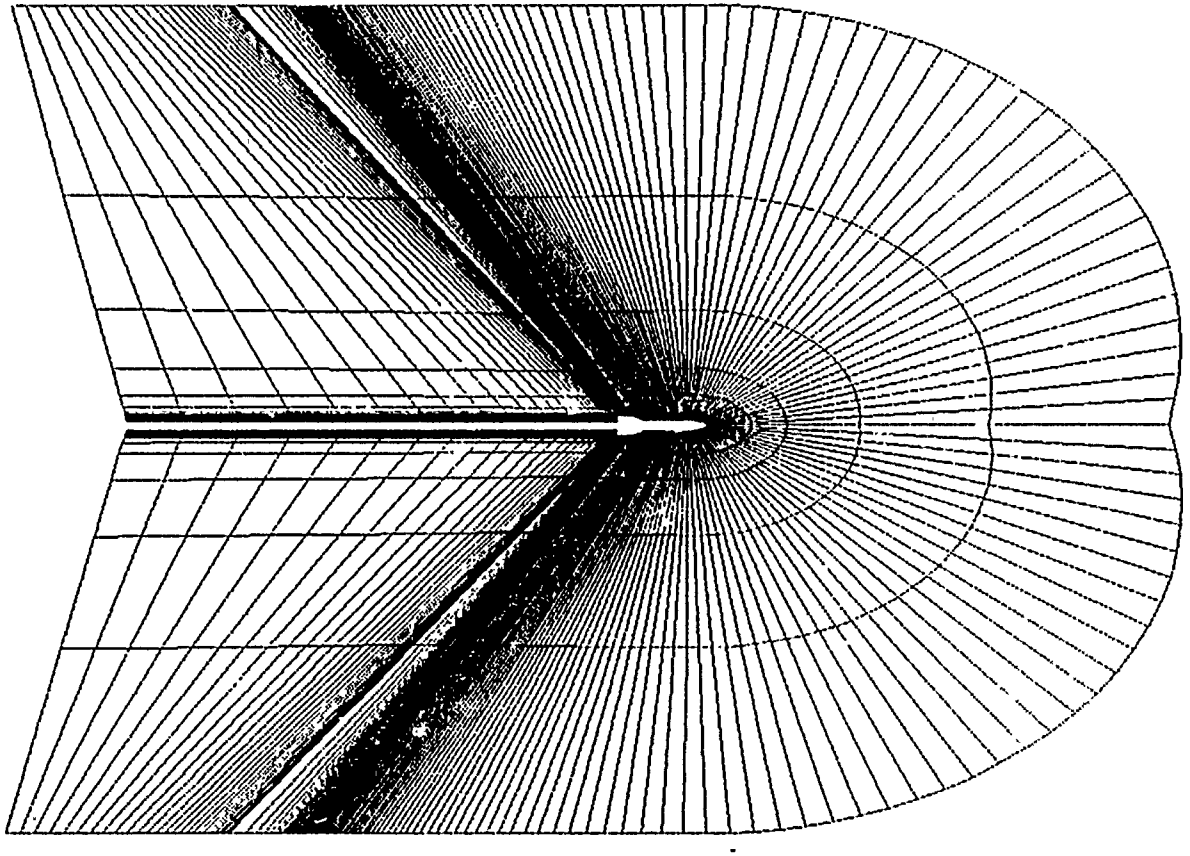

Figure 6a - Side view of Ogive-Cylinder-Ogive with fins field grid

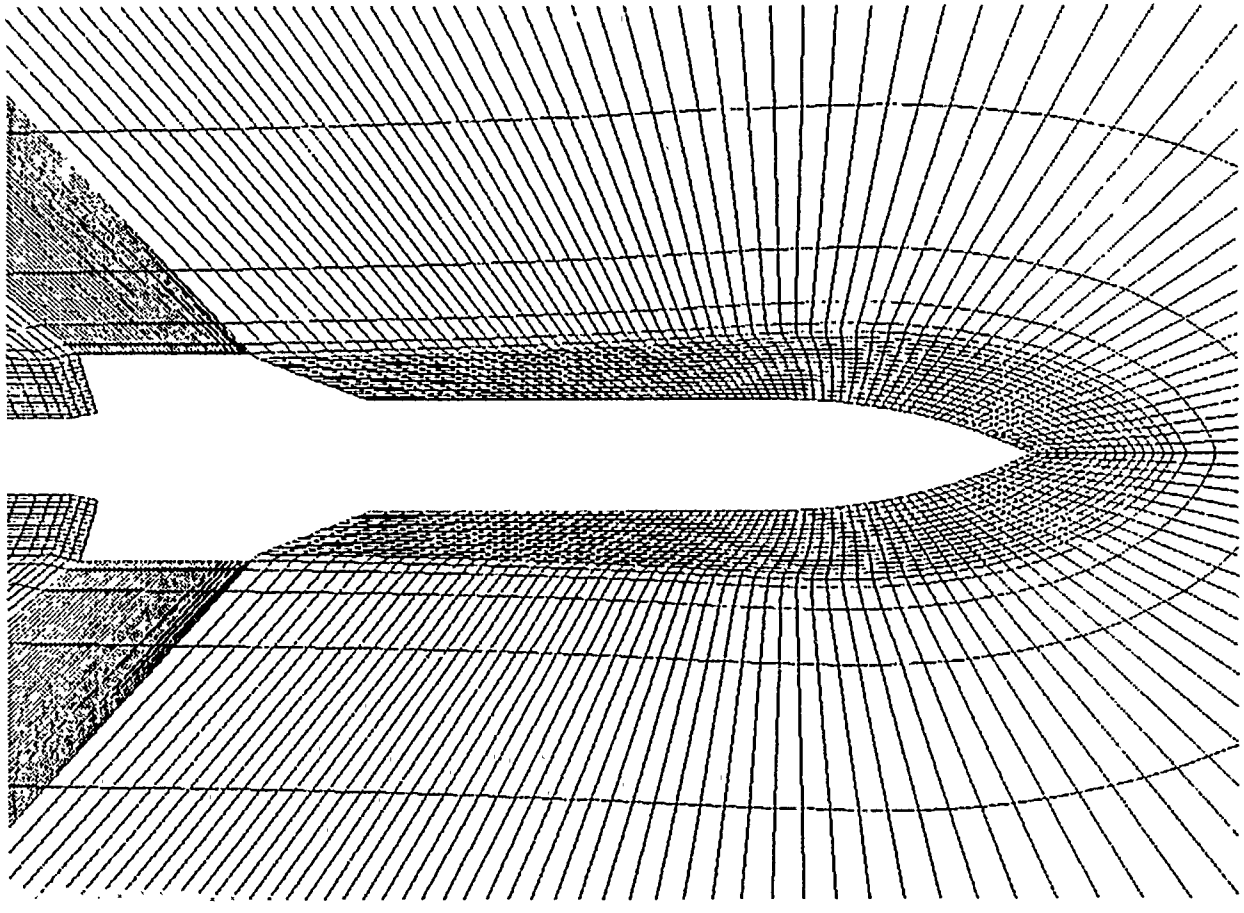

Figure 6b - Close-up side view of Ogive-Cylinder-Ogive with fins field grid

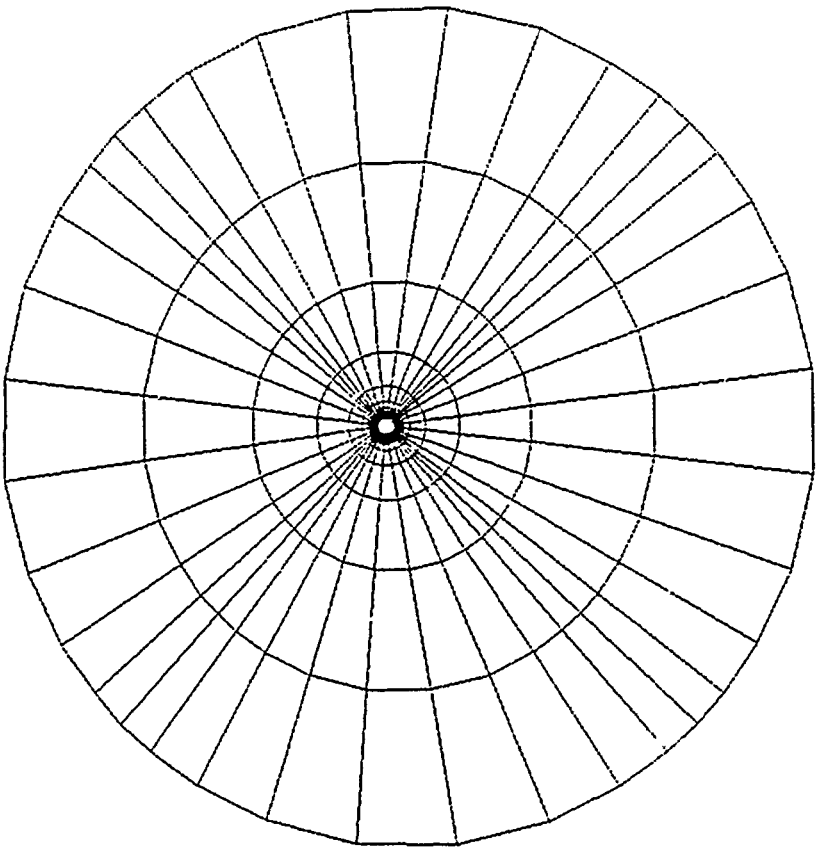

Figure 7a - Front view of Ogive-Cylinder-Ogive with fins field grid

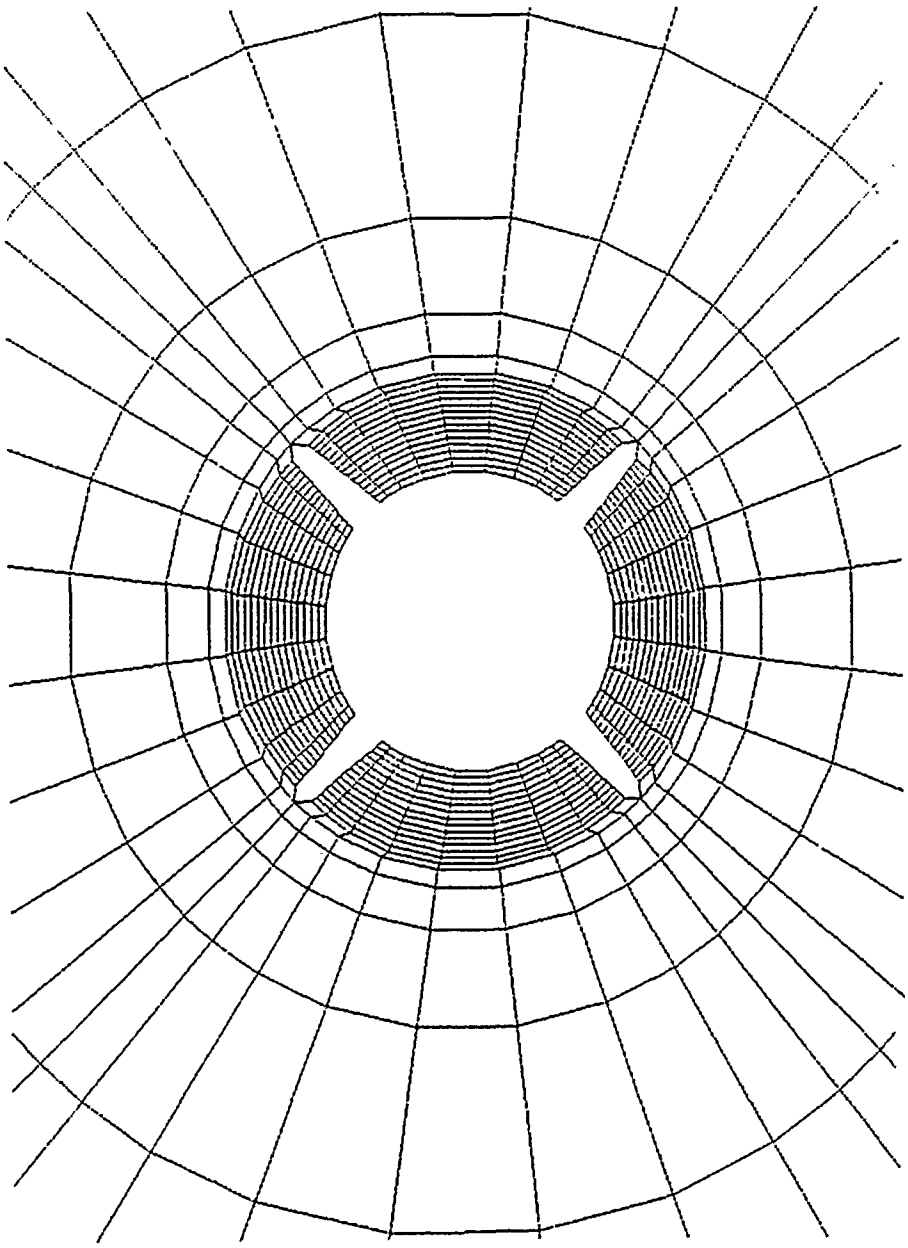

Figure 7b - Close-up front view of Ogive-Cylinder-Ogive with fins field grid

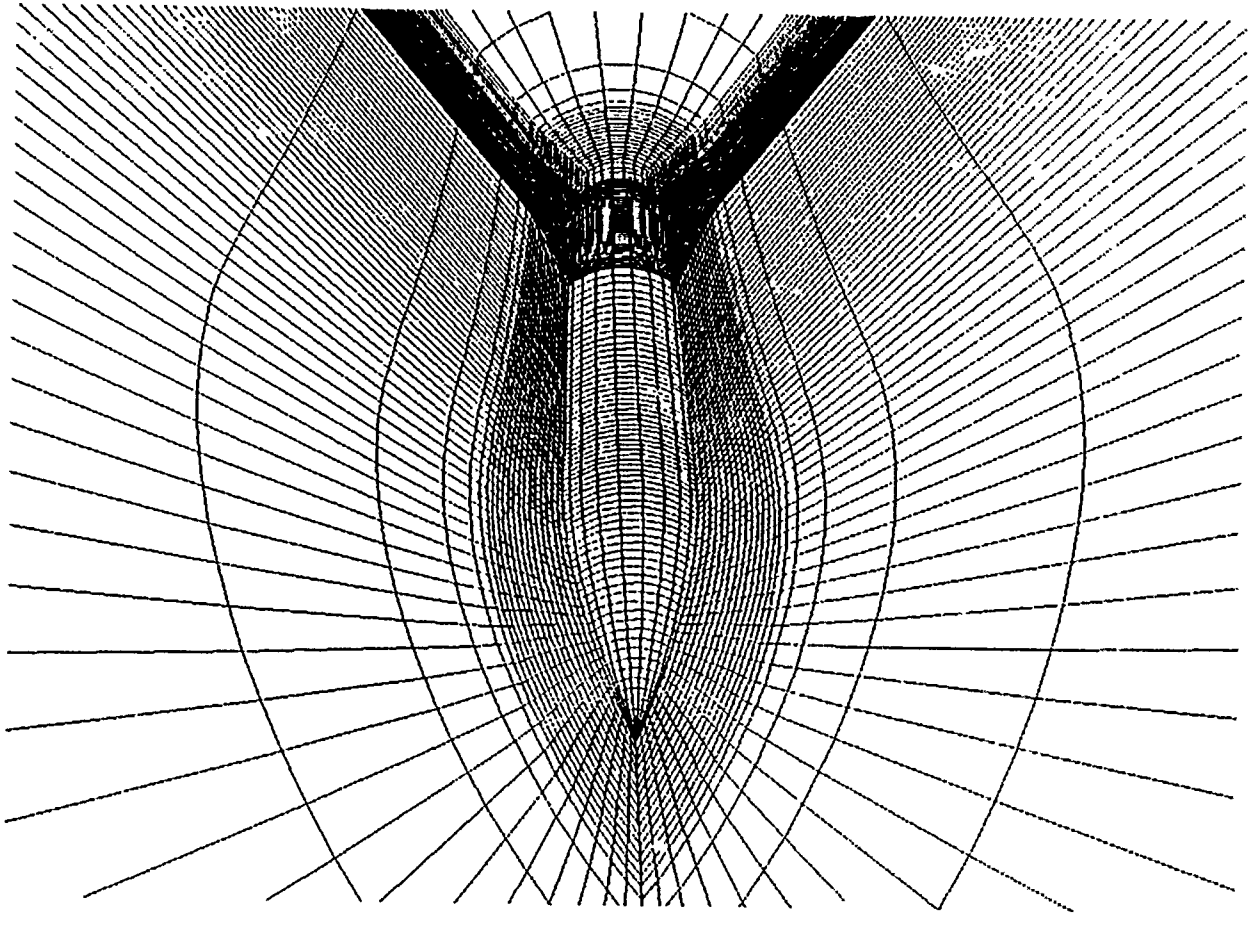

Figure 8 - Close-up perspective view of Ogive-Cylinder-Ogive with fins field grid

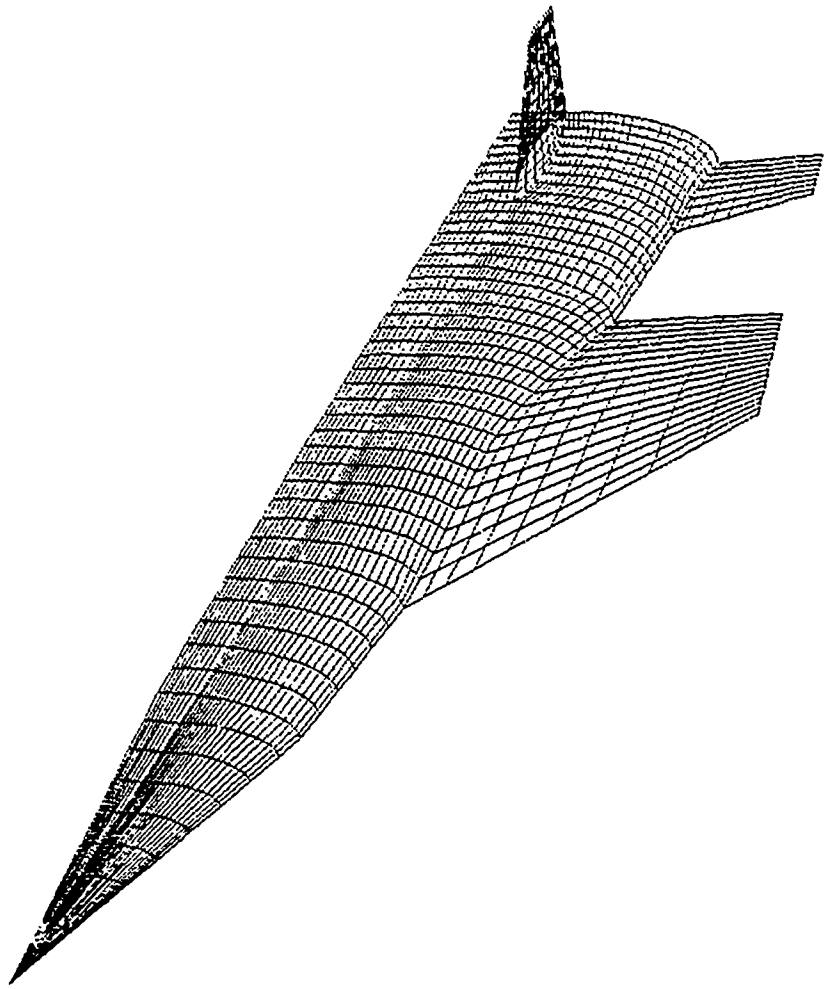

Figure 9 - Elliptic airframe with control surfaces surface grid

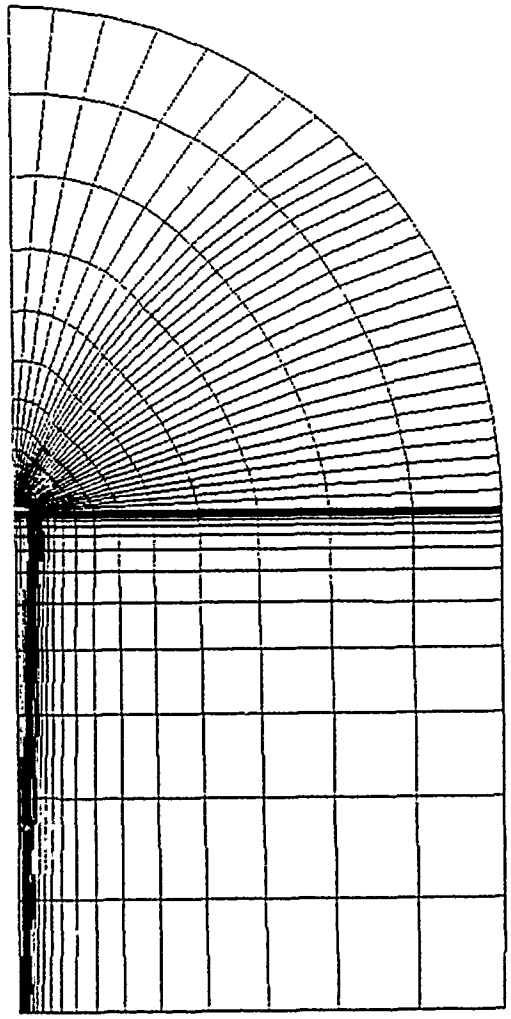

Figure 10a - Front view of elliptic airframe field grid

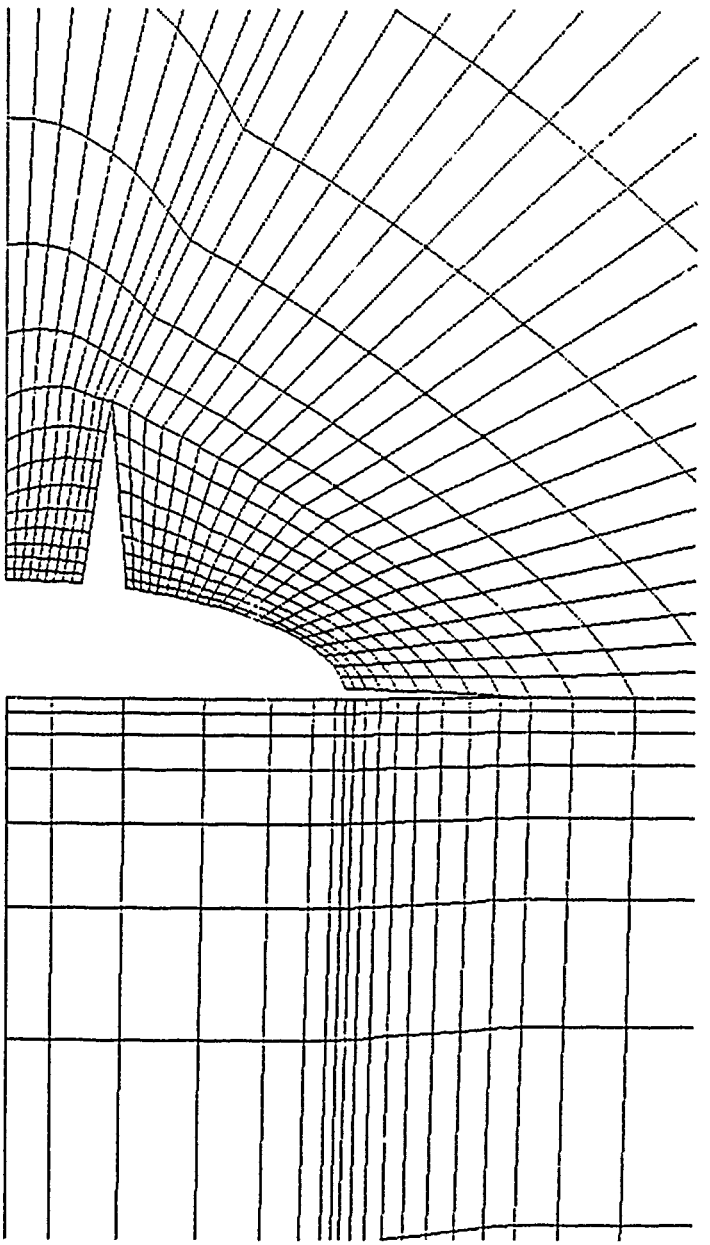

Figure 10b - Close-up front view of elliptic airframe field grid

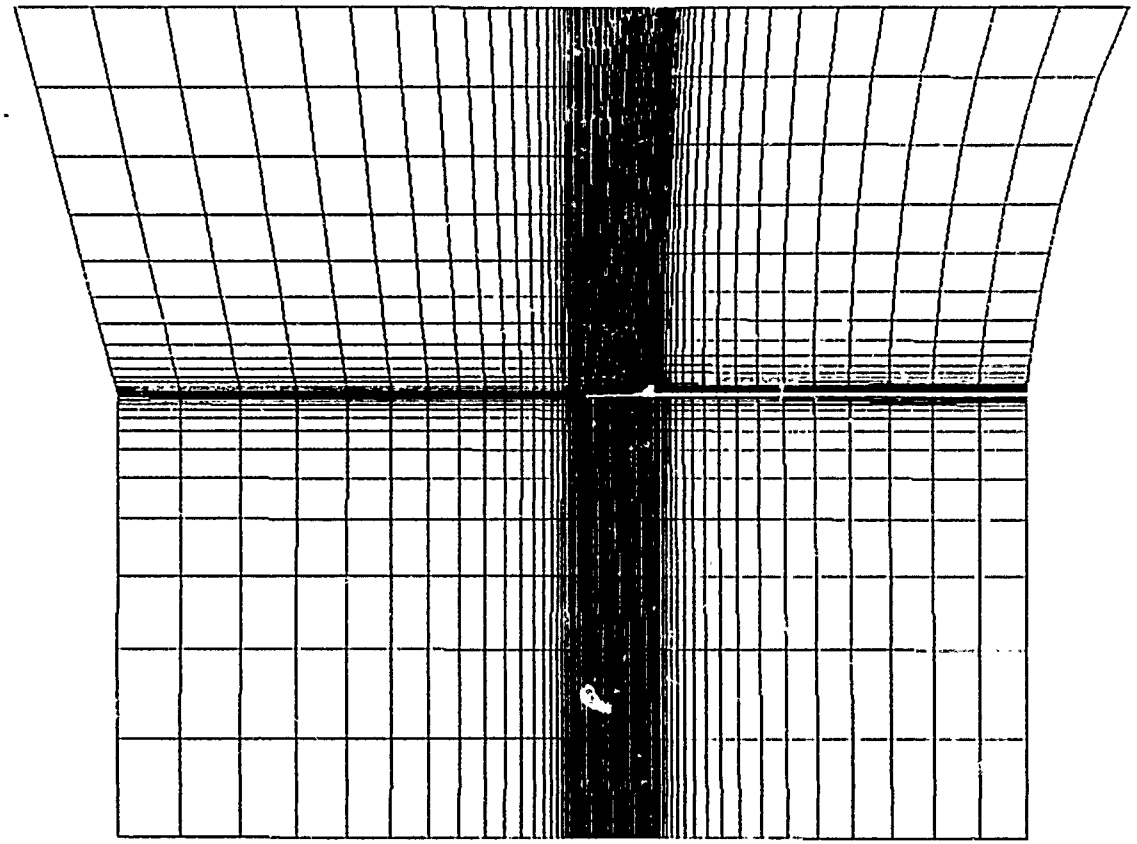

Figure 11a - Side view of elliptic airframe field grid

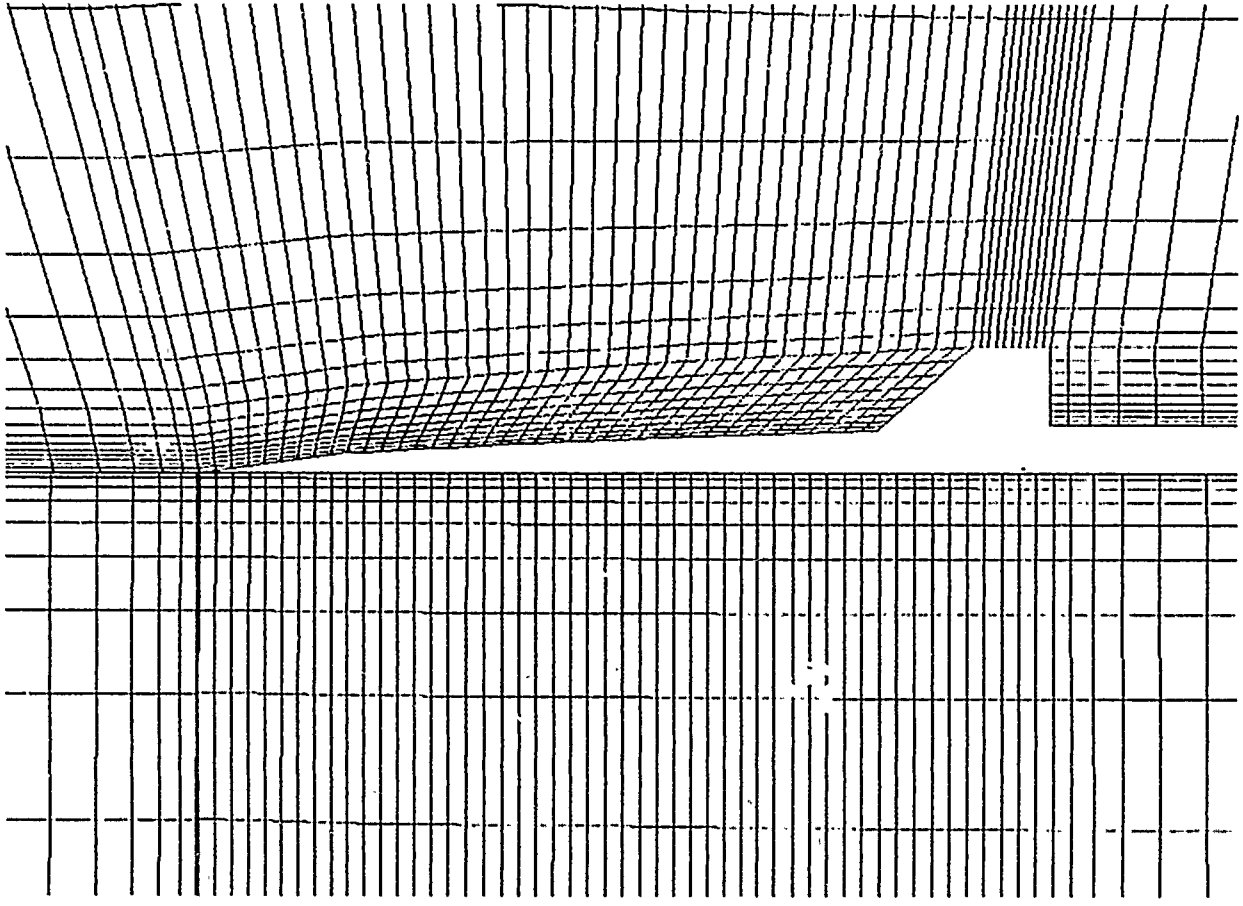

Figure 11b - Close-up side view of elliptic airframe field grid

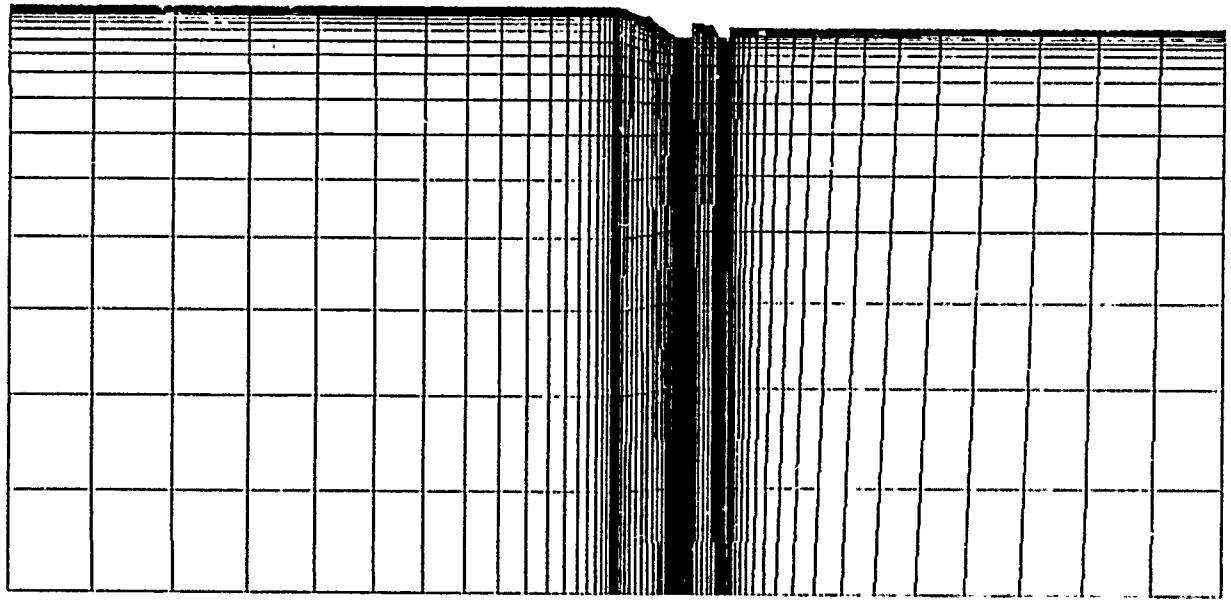

Figure 12a - Top view of elliptic airframe field grid

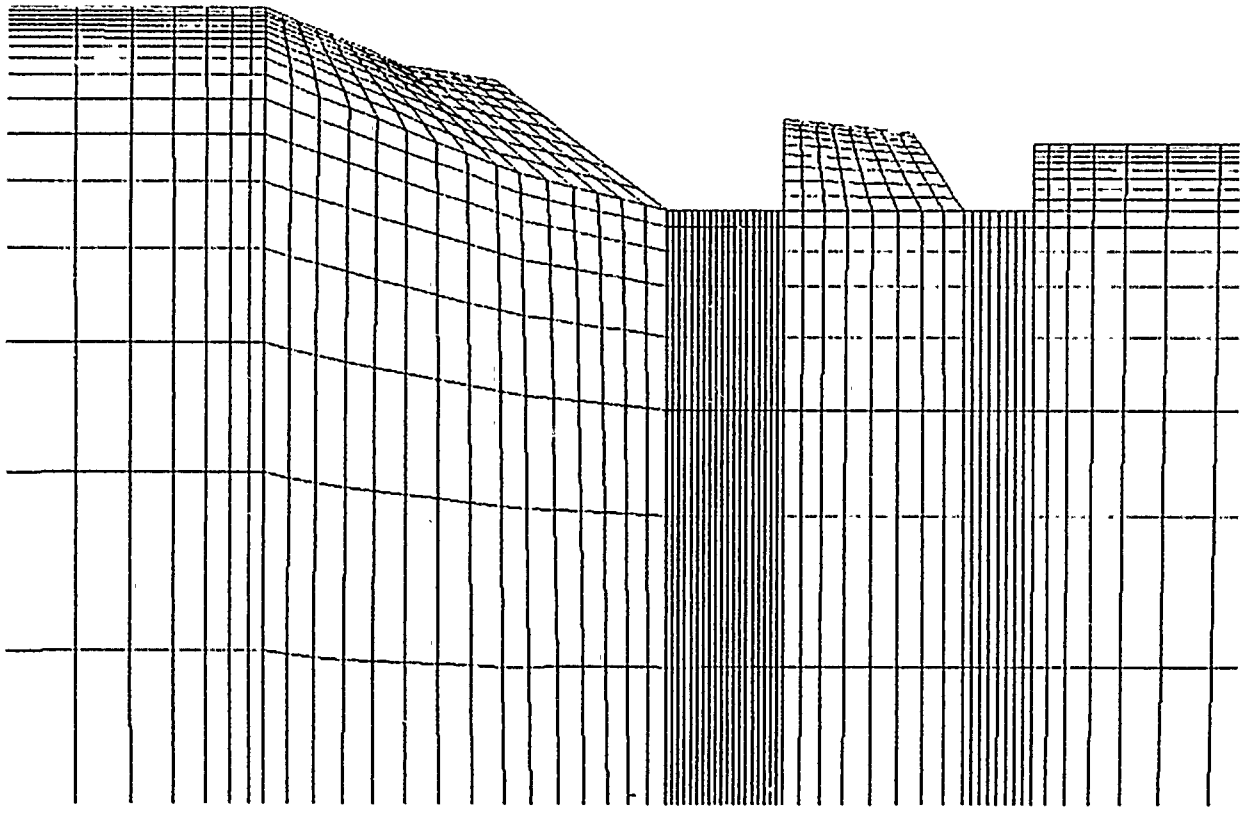

Figure 12b - Close-up top view of elliptic airframe field grid

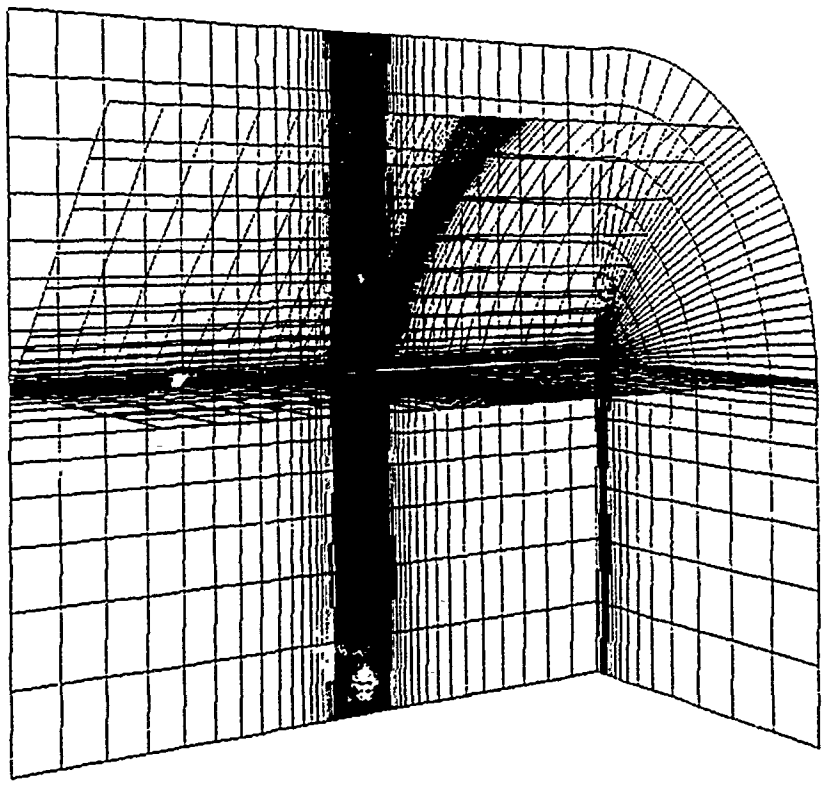

Figure 13a - Perspective view of elliptic airframe overall field grid

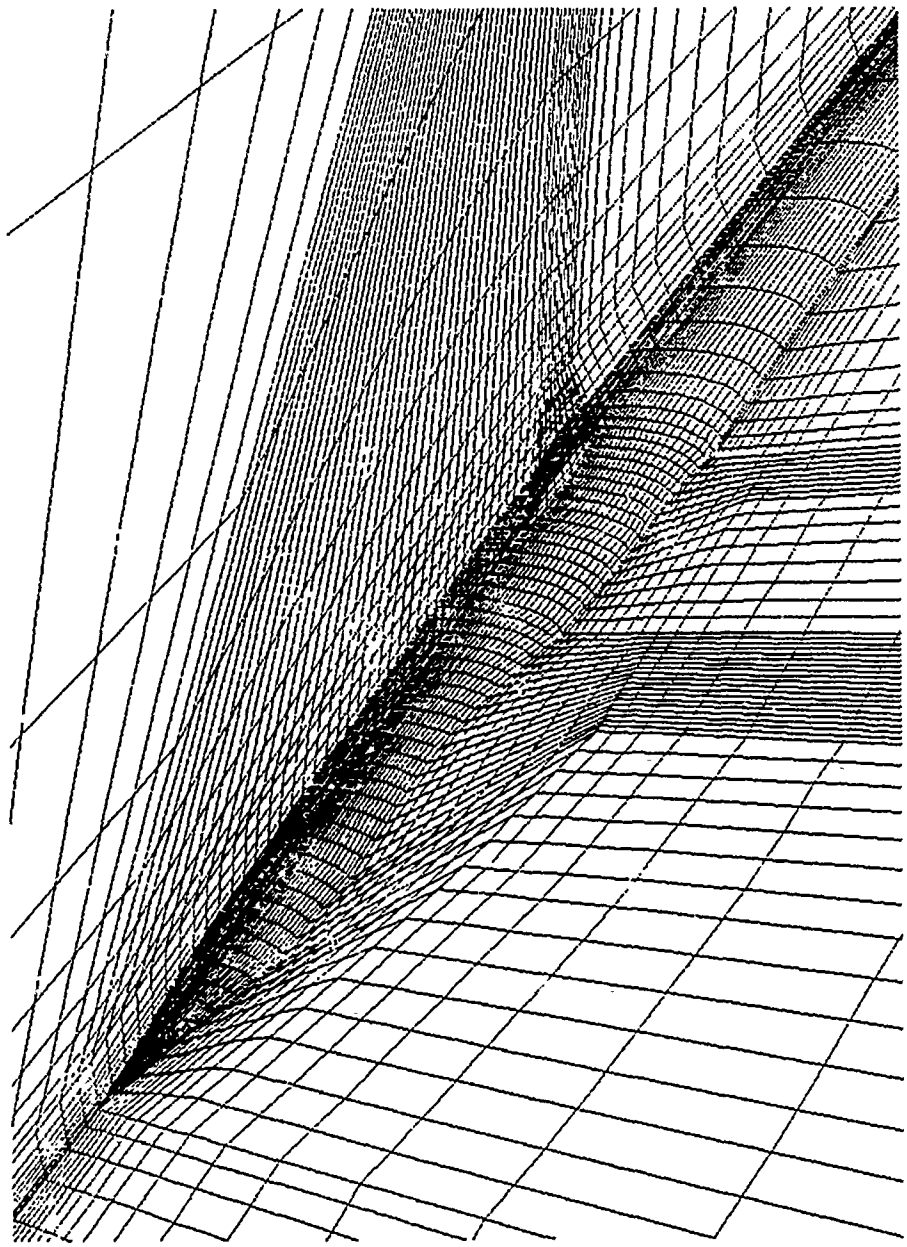

Figure 13b - Close-up perspective view of elliptic airframe overall field grid

Appendix A

SAMPLE PROGRAM SET-UP LISTING

```
      CHARACTER*8 NAMLST , NAMLIS , CDATA
      CHARACTER*144 STRING
      CHARACTER*64 ICHR
C
      INTEGER CPOS , IPOS , RPOS , PRMPT
C
      INTEGER  UP , DOWN , RSTART
C
      INTEGER  PLINES , PINL , PITEXT , PIECHO , PLS , PLT , PNL , PNJ ,
     &         PIEQUAL , PNUM
C
      PARAMETER( NCHAR = 31 , NINTG = 37 , NREALS =  5 )
C
      PARAMETER( NCVAL = 36, NIVAL = 2*NVALMX+41 , NRVAL = DIMP3 + 5 )
C
      PARAMETER( NIDAT =  3 , NRDAT =  2 )
C
      PARAMETER( NVAR = NCHAR + NINTG + NREALS )
C
      PARAMETER( NDAT = NIDAT + NRDAT )
C
      PARAMETER( IIP1 = NVALMX + 41 , ICP1 = 2 )
C
C.....................................................................
C
      PARAMETER  ( RNONE = 3.402824E+36 , INONE = 2147483647 )
C
C.....................................................................
C
      DIMENSION NAMLIS( NVAR ) , NDIM1( NVAR ) , NDIM2( NVAR ) ,
     &          NAMLST( 2 )
C
      DIMENSION ICHR( NCVAL ) , INTGR( NIVAL ) , REALS( NRVAL )
C
      DIMENSION CDATA( NDAT ) , IDAT( NIDAT ) , RDAT( NRDAT )
C
      DIMENSION CPOS( NCHAR ) , IPOS( NINTG ) , RPOS( NREALS )
C
CC          THIS SET FOR THE CHARACTER VARIABLES
C
      EQUIVALENCE ( ICHR( 1) ,    ITEM ) , ( ICHR( 2) ,     CLASS )
      EQUIVALENCE ( ICHR( 3) , INTERP ) , ( ICHR( 4) ,    PROTYP )
      EQUIVALENCE ( ICHR( 5) ,     FUN ) , ( ICHR( 6) ,     BLEND )
      EQUIVALENCE ( ICHR( 9) ,  REWIND ) , ( ICHR(10) ,    CONTYP )
      EQUIVALENCE ( ICHR(11) , RESTART ) , ( ICHR(12) ,     CHECK )
      EQUIVALENCE ( ICHR(13) ,  SMOCON ) , ( ICHR(14) ,       ALL )
      EQUIVALENCE ( ICHR(15) ,  WEIGHT ) , ( ICHR(16) ,      MATH )
      EQUIVALENCE ( ICHR(17) ,   OUTER ) , ( ICHR(18) ,    DFIRST )
      EQUIVALENCE ( ICHR(19) ,  DCROSS ) , ( ICHR(20) , PROFOR , PROF )
      EQUIVALENCE ( ICHR(23) ,  CONCUT ) , ( ICHR(24) ,    CONTIN )
      EQUIVALENCE ( ICHR(25) ,   TRIAD ) , ( ICHR(26) ,       NEW )
      EQUIVALENCE ( ICHR(27) ,  KSTORE ) , ( ICHR(28) ,      FORM )
      EQUIVALENCE ( ICHR(29) ,    MODE ) , ( ICHR(30) ,    IPRINT )
      EQUIVALENCE ( ICHR(31) ,    PART ) , ( ICHR(32) ,    BLKERR )
      EQUIVALENCE ( ICHR(33) ,  INTORT ) , ( ICHR(34) ,    FILNAM )
      EQUIVALENCE ( ICHR(35) ,  CONTUR )
C
CC        THE NEXT SET IS FOR THE INTEGER VARIABLES.
C
      EQUIVALENCE ( INTGR( 1) ,   START ,  POINT )
      EQUIVALENCE ( INTGR( 4) ,  ISTART , IPOINT )
```

```
      EQUIVALENCE ( INTGR( 7) ,     END ) , ( INTGR(10) ,    IEND )
      EQUIVALENCE ( INTGR(13) ,   BLOCK ,  B , BLOC )
      EQUIVALENCE ( INTGR(14) ,  IBLOCK , IB )
      EQUIVALENCE ( INTGR(15) ,   ORDER , SMODIR ,  S )
      EQUIVALENCE ( INTGR(18) ,  RORDER , SMOCOM , RS )
      EQUIVALENCE ( INTGR(21) ,  OPOINT ) , ( INTGR(24) ,   ITMAX )
      EQUIVALENCE ( INTGR(25) ,    FILE ) , ( INTGR(26) ,  CONUPI )
      EQUIVALENCE ( INTGR(27) , VALOUT,VALUE ) , ( INTGR(28) , LOCAT )
      EQUIVALENCE ( INTGR(31) , SEGMENT ) , ( INTGR(32) ,  POINTS )
      EQUIVALENCE ( INTGR(33) ,  DIRECT ) , ( INTGR(34) ,    NDEX )
      EQUIVALENCE ( INTGR(35) ,  PRODIR , PROJ )
      EQUIVALENCE ( INTGR(38) ,    SIZE )
      EQUIVALENCE ( INTGR(41) ,   TERMS ) , ( INTGR(IIP1) ,ITERMS )
C
C           THE NEXT SET IS FOR THE REAL VARIABLES.
C
      EQUIVALENCE ( REALS( 1) ,  ACCPAR ) , ( REALS( 2) ,     TOL )
      EQUIVALENCE ( REALS( 3) ,  CONFAC ) , ( REALS( 4) ,  SPAVAL )
      EQUIVALENCE ( REALS( 5) ,  VALUES )
C
C     THIS SET OF EQUIVALENCE STATEMENTS MAINLY USED WHEN CHARACTER
C         VALUES ARE INPUTS TO INTEGER OR REAL VARIABLES.
C
      EQUIVALENCE( IDAT(1) ,      UP ) , ( IDAT(2) ,    DOWN )
      EQUIVALENCE( IDAT(3) ,  RSTART )
C
      EQUIVALENCE( RDAT(1) , OPTIMUM ) , ( RDAT(2) ,  EXTRAP )
C
C-----------------------------------------------------------------------
C
C       THIS SET OF STATEMENTS ARE FOR QUANTITIES OF TYPE CHARACTER
C
      DATA NAMLST/ 'GINPUT' , 'GOUTPUT' /
C
      DATA (NAMLIS(I),I=1,NCHAR)/ 'ITEM', 'CLASS' , 'INTERP'  ,
     &            'PROTYP' ,     'FUN' ,   'BLEND' ,  'REWIND' ,
     &            'CONTYP' , 'RESTART' ,   'CHECK' ,  'SMOCON' ,
     &               'ALL' ,  'WEIGHT' ,    'MATH' ,   'OUTER' ,
     &            'DFIRST' ,  'DCROSS' ,  'PROFOR' ,  'CONCUT' ,
     &            'CONTIN' ,   'TRIAD' ,     'NEW' ,  'KSTORE' ,
     &              'FORM' ,    'MODE' ,  'IPRINT' ,    'PART' ,
     &            'BLKERR' ,  'INTORT' ,  'FILNAM' ,  'CONTUR' /
C
C       THIS SET OF STATEMENTS ARE FOR QUANTITIES OF TYPE INTEGER
C
      DATA (NAMLIS(I),I=NCHAR+1,NINTG+NCHAR) /
     &             'START' ,  'ISTART' ,     'END' ,    'IEND' ,
     &             'BLOCK' ,  'IBLOCK' ,   'ORDER' ,  'SMODIR' ,
     &            'RORDER' ,  'SMOCOM' ,   'POINT' ,  'IPOINT' ,
     &            'OPOINT' ,   'ITMAX' ,    'FILE' ,  'CONUPI' ,
     &            'VALOUT' ,   'LOCAT' , 'SEGMENT' ,  'POINTS' ,
     &            'DIRECT' ,    'NDEX' ,  'PRODIR' ,    'SIZE' ,
     &             'TERMS' ,  'ITERMS' ,   'VALUE' /
C
C       THIS SET OF TSTATEMENTS ARE FOR QUANTITIES OF TYPE REAL
C
      DATA (NAMLIS(I),I=NINTG+NCHAR+1,NCHAR+NINTG+NREALS) /
     &            'ACCPAR' ,     'TOL' ,  'CONFAC' ,  'SPAVAL' ,
     &            'VALUES' /
C
C THIS FIRST SET OF DATA VALUES FOR THE ARRAY NDIM IS FOR CHARACTER
```

```
C     VARIABLES. ANY ADDITIONS OF CHARACTER VARIABLES SHOULD COME AT THE
C     END. IN THE EVENT AN ARRAY OF VARIABLE SIZE IS TO BE ADDED MAKE
C     SURE THOSE ARRAY SIZES ARE ALWAYS AT THE END.
C
C   THE SECOND SET IS FOR INTEGERS. NOTE THAT THE VARIABLE DIMENSIONED
C     ARRAY SIZES ARE AT THE END OF THE INTEGER LIST.
C
C   THE LAST GROUP OF VALUES ARE FOR REAL VARIABLES. AGAIN, NOTE THAT
C     THE VARIABLE DIMENSIONED ARRAY SIZES ARE AT THE END OF THE REAL
C     LIST.
C
      DATA NDIM1/  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     &             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
C
     &            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     &            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
C
     &            1, 1, 1, 1, 1 /
C
      DATA NDIM2/  1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,
     &             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ICP1,
C
     &            3, 3, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1,
     &            1, 3, 1, 1, 1, 1, 3, 3, NVALMX,NVALMX, 1,
C
     &            1, 1, 1, 1,DIMP3/
C
C   THESE ARRAYS USED BY THE PARSER FOR POINTING TO A MEMORY LOCATION.
C
      DATA CPOS/ 1, 2, 3, 4, 5, 6, 9,10,11,12,13,14,15,16,17,18,19,20,
     &          23,24,25,26,27,28,29,30,31,32,33,34,35/
C
      DATA IPOS/ 1, 4, 7,10,13,14,15,15,18,18, 1, 4,21,24,25,26,27,28,
     &          31,32,33,34,35,38,41,IIP1,27/
C
      DATA RPOS/ 1, 2, 3, 4, 5/
C
C.....................................................................
C
C     THESE STATEMENTS ARE ADDITIONS. 070988 <sus AM>
C
      DATA  UP/ 2147483644 / , DOWN/ 2147483645 /
C
      DATA RSTART/ 2147483646 /
C
      DATA OPTIMUM/2147483647.1/ , EXTRAP/2147483647.1/
C
      DATA CDATA/ 'UP' , 'DOWN' , 'RESTART' , 'OPTIMUM' , 'EXTRAP' /
C
C.....................................................................
C
      DATA PLINES/ 1 / , PIECHO/ 0 / , PINL/ 1 / , PLS/ 1 / , PLT/ 1 / ,
     &     PNL/ 1 / , PNJ/ 1 /
C
C**********************************************************************
C
                              :
                              :
                              :
                              :
C
```

```
C
C********************* READ NAMELIST ***********************************
C
C THE FOLLOWING LINES REPLACE THE LINE 'READ INPUT'. THESE LINES CALL
C   THE NAMELIST INPUT EMULATOR. NOTE THAT THE ROUTINE 'PROMPT'
C   SETS UP A PROMPT FOR THE INTERACTIVE USER.
C
      PRMPT = PRMPT + 1
      NDS = 0
C
      IF( IBATCH .EQ. 0 ) CALL PROMPT( PRMPT )
C
C NOW READ IN THE INPUT STRING AND DETERMINE IF ITS TOO LONG.
C
  199 CONTINUE
C
      CALL RDSTRG( STRING , LCT , READFL , NSTRNG , ISTRNG , ISTAT )
C
      IF( ISTAT .EQ. 1 ) GO TO 100
C
C  PASS THE CHARATER STRING 'STRING' TO THE PARSER AND PARSE THE INPUTS.
C
      CALL PARSER( STRING,NAMLST(1),NAMLIS,NDIM1,NDIM2,ICHR,INTGR,
     &             REALS,CDATA,IDAT,RDAT , CPOS , IPOS , RPOS , NVAR ,
     &             NCHAR , NINTG , NREALS , NCVAL , NIVAL , NRVAL ,
     &             NDAT , NIDAT , NRDAT , NDS , PLINES , PINL ,
     &             PITEXT , PIECHO , PLS , PLT , PNL , PNJ , PIEQUAL ,
     &             PNUM , ISTAT )
C
      IF ( ISTAT .EQ. 1 ) GO TO 100
C
C*************************************************************************
C
C
      IF( NDS .EQ. 0 ) GO TO 199
```

Appendix B

GENERALIZED PARSER/NAMELIST INPUT EMULATOR LISTING

```
      SUBROUTINE RDSTRG( STRING , LCT , IREAD , NSTRNG , ISTRNG , ISTAT)
      CHARACTER*(*) STRING
C
      IF( IREAD .EQ. 1 ) THEN
        READ( 2,999 ) STRING
      ELSE
        READ( 5,999 ) STRING
      ENDIF
C
      LCT = ILEN( STRING , ISTRNG )
C
      IF( LCT .GT. NSTRNG ) THEN
         LST = LCT - NSTRNG
         PRINT *,' ERROR - TOO MANY CHARACTERS IN THIS LINE '
         PRINT *, STRING( 1:LCT )
         PRINT *,' SHORTEN LINE BY APPROXIMATLY ',LST,' CHARACTERS.'
         PRINT *,' '
         ISTAT = 1
         RETURN
      ENDIF
C
 999  FORMAT(A)
C
      RETURN
      END
```

```
      SUBROUTINE PARSER( STRING,NAMLST,NAMLIS,NDIM1,NDIM2,ICHR , INTGR,
     &                   REALS,CDATA , IDAT , RDAT , CPOS , IPOS , RPOS,
     &                   NVAR , NCHAR , NINTG , NREALS , NCVAL , NIVAL ,
     &                   NRVAL , NDAT , NIDAT , NRDAT , NDS , L , INL ,
     &                   ITEXT , IECHO , LS , LT , NL , NJ , IEQUAL ,
     &                   NUM , ISTAT )
C
C*********************************************************************
C
C  THESE STATEMENTS USED FOR PARSING. 070988 <sus AM>
C   THE FOLLOWING IS A DESCRIPTION OF THE VARIABLES USED FOR PARSING:
C
C         STRING - INPUT CARACTER LINE;PASSED TO THE PARSER FOR
C                  PARSING.
C
C         NAMLST - CONTAINS THE NAME OF THE NAMELIST.
C
C         NAMLIS - CHARACTER ARRAY DIMENSIONED TO THE NUMBER OF
C                  VARIABLES IN THE NAMELIST ( NVAR ) AND WHICH
C                  CONTAINS THE LIST OF INPUT ITEMS IN NAMELIST.
C                  THIS ARRAY IS ORGANIZED WITH INPUT ITEMS OF TYPE
C                  CHARACTER FIRST , INTEGER SECOND , AND REALS LAST.
C
C           ICHR - CHARACTER ARRAY WHICH HOLDS THE VALUES FOR THE
C                  INPUT ITEMS OF TYPE CHARACTER. THIS ARRAY IS THEN
C                  EQUIVALENCED TO THE APPROPRIATE VARIABLES. THIS
C                  ARRAY IS DIMENSIONED TO THE NUMBER OF CHARACTER
C                  VALUES ( NCVAL ) .
C
C           INTGR - INTEGER ARRAY WHICH HOLDS INTEGER VALUES FOR THE
C                  INPUT ITEMS OF TYPE INTEGER. THIS ARRAY IS THEN
C                  EQUIVALENCED TO THE PROPER INTEGER VARIABLES.
C                  INTGR IS DIMENSIONED TO THE NUMBER OF INTEGER
C                  VALUES ( NIVAL ) IN NAMLIS.
C
C           REALS - REAL ARRAY WHICH HOLDS REAL VALUES FOR THE INPUT
C                  ITEMS OF TYPE REAL. IT THEN IS EQUIVALENCED TO THE
C                  APPROPRIATE VARIABLES. THE ARRAY REAL IS DIMENSIONED
C                  TO THE NUMBER OF REAL VALUES ( NRVAL ) IN NAMLIS.
C
C  NDIM1 & NDIM2 - ARRAYS CONTAINING THE LOWER & UPPER BOUNDS ,
C                  RESPECTIVELY, OF EACH OF THE VARIABLES IN NAMLIS IN
C                  THE ORDER OF CHARACTER FIRST, INTEGER SECOND , AND
C                  REAL AS LAST. NDIM1 AND NDIM2 ARE DIMENSIONED BY THE
C                  THE TOTAL NUMBER OF VARIABLES IN NAMELIS( NVAR ).
C
C           NCHAR - NUMBER OF CHARACTER VARIABLES
C
C           NINTG - NUMBER OF INTEGER VARIABLES
C
C          NREALS - NUMBER OF REAL VARIABLES
C
C                >>>NOTE: NVAR = NCHAR + NINTG + NREALS<<<<<<<<
C
C
C   THE FOLLOWING ARRAYS ARE USED WHEN CHARACTER INFORMATION IS PASSED
C      IN INTEGER OR REAL VARIABLES( IN THE INPUT ) INSTEAD OF INTEGER
C      OR REAL VALUES. THIS IS USEFUL IN THAT IT ALLOWS THE USER SOME
C      FLEXIBILITY,i.e., THE USER DOES NOT HAVE TO SUPPLY THE APPRO-
C      PRIATE VALUES ( INTEGER OR REAL ) BUT CAN INPUT SAY 'FIRST',
C      'LAST', OR 'SAME'( OR WHATEVER, AS DESCRIBED BY THE USERS MANUAL -
```

```
C       'PROGRAM EAGLE - USERS MANUAL VOLUME II' BY DR. JOE F. THOMPSON
C        MISS. STATE UNIV , DEPT OF AEROSPACE ENGINEERING , MS ).
C
C
C          CDATA - CHARACTER ARRAY CONTAINING THE NAMES OF CONSTANTS.
C                  THIS ARRAY IS DIMENSIONED TO THE NUMBER OF
C                  INTEGER AND REAL CONSTANTS( NDAT ).
C
C           IDAT - INTEGER ARRAY CONTAINING INTEGER CONSTANTS
C                   INITIALIZED BY THE MAIN PROGRAM. THE ARRAY IS
C                   EQUIVALENCED TO THE APPROPRIATE VALUES. IDAT IS
C                   DIMENSIONED BY THE NUMBER OF INTEGER VALUES IN
C                   'COMMON/ CONST / ...'( NIDAT ).
C
C           RDAT - REAL ARRAY CONTAINING INTEGER CONSTANTS INITIALIZED
C                   IN THE MAIN PROGRAM. THIS ARRAY IS EQUIVALENCED TO
C                   THE APPROPRAITE VALUES AND IS DIMENSIONED TO THE
C                   NUMBER OF REAL VALUES IN 'COMMON/ CONST / ...'
C                   ( NRDAT ).
C       >>>>>>>NOTE: - NDAT = NIDAT + NRDAT <<<<<<<<<<<<<<<<<<<<<<<<<<
C
C           CPOS - INTEGER ARRAY WHICH HOLDS THE ACTUAL ARRAY POSITIONS
C                  OF ALL THE NAMELIST INPUT VARIABLES OF TYPE
C                  CHARACTER. CPOS ACTS AS A POINTER FOR THE PARSER.
C
C           IPOS - INTEGER ARRAY WHICH HOLDS THE AC AL ARRAY POSITIONS
C                  OF ALL THE NAMELIST INPUT VARIABLES OF TYPE INTEGER.
C                  IPOS ACTS AS A POINTER FOR THE PARSER.
C
C           RPOS - INTEGER ARRAY WHICH HOLDS THE ACTUAL ARRAY POSITIONS
C                  OF ALL THE NAMELIST INPUT VARIABLES OF TYPE REAL.
C                  RPOS ACTS AS A POINTER FOR THE PARSER.
C
C.....................................................................
C
C       OTHER ADDITIONS:
C
C              ISTAT - STATUS FLAG. 'ISTAT = 0' NO ERRORS;'ISTAT = 1'
C                      INDICATES AN ERROR. IT WILL PROMPT THE USER IN
C                      INTERACTIVE MODE.OTHERWISE, EXECUTION WILL STOP.
C
C**********************************************************************
C
C**********************************************************************
C
C
      CHARACTER*(*) NAMLST , NAMLIS , CDATA
      CHARACTER*144 TEXT
      CHARACTER*(*) STRING
      CHARACTER*(*) ICHR
C
      INTEGER CPOS , IPOS , RPOS , POINTR
C
      PARAMETER( NTEXT = 144 )
C
      DIMENSION ICHR( NCVAL ) , INTGR( NIVAL  ) , REALS( NRVAL )
      DIMENSION NAMLIS( NVAR ) , NDIM1( NVAR ) , NDIM2( NVAR )
      DIMENSION CDATA( NDAT ) , IDAT( NIDAT ) , RDAT( NRDAT )
      DIMENSION CPOS( NCHAR ) , IPOS( NINTG ) , RPOS( NREALS )
C
C*********/**********************************************************
```

```
C*************************************************************************
C
      ICO = ICHAR( '0' )
      IC9 = ICHAR( '9' )
      ICA = ICHAR( 'A' )
      ICZ = ICHAR( 'Z' )
      ICQU = ICHAR( '''' )
      ICCM = ICHAR( ',' )
      ICPOS = ICHAR( '+' )
      ICMIN = ICHAR( '-' )
      ICPER = ICHAR( '.' )
C
      IPAREN = 0
      IPOINT = 1
C
      LNL = ILEN( NAMLST 8 )
       NL = LNL + 2
C
      DO 1 I = 1,NTEXT
 1      TEXT( I:I ) = ' '
C
      ISTRNG = LEN( STRING )
      LC = ILEN( STRING , ISTRNG )
C
      IF( STRING( 1:1 ) .NE. ' ' ) THEN
C
         IF( STRING( 1:1 ) .EQ. 'C' ) THEN
C
           PRINT *, STRING( 1:LC )
           PRINT *,' '
           GO TO 9999
C
         ELSE IF( STRING( 1:1 ) .EQ. 'E' .AND. L .EQ. 1 ) THEN
C
           PRINT *, STRING( 1:LC )
           PRINT *,' '
           IECHO = 1
C
         ELSE IF( L .EQ. 1 ) THEN
C
           PRINT *,' ERROR - COLUMN 1 MUST EITHER BE BLANK, HAVE A ',
     &               '<C> OR AN <E> '
           PRINT *,' '
           ISTAT = 1
           GO TO 9999
C
         ELSE
C
           PRINT *,' ERROR - COLUMN 1 MUST BE BLANK '
           PRINT *,' '
           ISTAT = 1
           GO TO 9999
C
         ENDIF
C
      ELSE IF( STRING( 1:1 ) .EQ. ' ' .AND. IECHO .EQ. 1 ) THEN
C
           PRINT *, STRING( 1:LC )
           PRINT *,' '
C
      ENDIF
```

```
C
      IF( L .EQ. 1 ) THEN
C
        IF( STRING( 2:2 ) .NE. '$' ) THEN
          PRINT *,' ERROR - DOLLAR SIGN REQUIRED IN COLUMN 2 '
          PRINT *,' '
          ISTAT = 1
          GO TO 9999
        ENDIF
C
        NDS = INDEX( STRING( 3:LC ) , '$' )
C
        IF( NDS .NE. 0 )  NDS = NDS + 2
C
      ELSE
C
        NDS = INDEX( STRING( 2:LC ) , '$' )
        IF( NDS .NE. 0 )  NDS = NDS + 1
C
      ENDIF
C
      IF( NDS .EQ. 0 ) THEN
C
        LS = LC
C
      ELSE
C
        LS = NDS
C
      ENDIF
C
C  TEST TO SEE IF THE PROPER NAME OF THE NAMELIST IS FOUND IN THE
C     STRING IN THE RIGHT PLACE.
C
      IF( L .EQ. 1 ) THEN
C
      IF( STRING( 3:NL ) .NE. NAMLST ) THEN
        PRINT *,' ERROR - ',NAMLST,' NOT FOUND BETWEEN COLUMNS 3 THRU',
     &            NL,' . ', STRING( 1:NL )
        PRINT *,' '
        ISTAT = 1
        GO TO 9999
      ENDIF
C
C  TEST TO SEE IF A BLANK EXISTS AFTER NAMLST IN STRING.
C
      IF( STRING( NL+1:NL+1 ) .NE. ' ' ) THEN
        PRINT *,' ERROR - COLUMN AFTER ',NAMLST,' MUST BE BLANK .',
     &            STRING( 1:NL+1 )
        PRINT *,' '
        ISTAT = 1
        GO TO 9999
      ENDIF
C
C  SET STRING POINTER( NL ) TO PROPER POSITION FOR THE PARSER .
C
      NL = NL + 2
C
      ELSE
C
      NL = 2
```

```
C
      ENDIF
C
 10   CONTINUE
C
C
C  THIS SECTION PUTS TOGETHER A STRING OF CHARACTERS TO FORM A
C     QUANTITY TO BE LATER DETERMINED IF THE QUANTITY IS IN THE
C     LIST NAMLIS.
C
      DO 20 I = NL,LS
C
C  TEST TO SEE IF ANY LEADING BLANKS EXIST;IF SO INCREMENT
C
        IF( STRING( I:I ) .EQ. ' ' .AND. TEXT( 1:1 ) .EQ. ' ' ) GO TO 20
C
        IF( STRING(I:I) .EQ. '$' .AND. I .EQ. LS ) THEN
C
          IF( TEXT(1:1) .NE. ' ' ) THEN
            PRINT *,' ERROR - NOT FINISH READING THE QUANTITY WHEN ',
     &              'INPUT LINE WAS TERMINATED ',STRING( LS-7:LS )
            ISTAT = 1
            GO TO 9999
          ELSE
            GO TO 9999
          ENDIF
C
        ENDIF
C
        ICHS = ICHAR( STRING( I:I ) )
C
C  TEST TO SEE IF THE CHARACTER STRING STARTS WITH AN INTEGER, IF SO
C     THEN ITS AN ERROR.
C
        IF( L .EQ. 1 ) THEN
C
        IF((ICHS .GE. IC0 .AND. ICHS .LE. IC9) .AND. TEXT(1:1) .EQ. ' ')
     &                                                          THEN
          PRINT *,'ERROR - QUANTITY CANNOT START WITH A NUMBER ',
     &                                              STRING( I:I+7 )
          PRINT *,' '
          ISTAT = 1
          GO TO 9999
        ENDIF
C
        ELSE IF((( ICHS .GE. IC0 .AND. ICHS .LE. IC9 ) .OR.
     &          ICHS .EQ. ICQU .OR. ICHS .EQ. ICCM .OR.
     &          ICHS .EQ. ICPOS .OR. ICHS .EQ. ICMIN .OR.
     &          ICHS .EQ. ICPER ) .AND. TEXT(1:1) .EQ. ' ' ) THEN
C
                IEQUAL = 1
                NL = I
                GO TO 210
C
        ENDIF
C
C  FIRST TEST TO SEE IF THE CHARACTER STRING( I:I ) IS BETWEEN 0->9
C    AND A->Z IF SO CONCATENATE. IF NOT THEN TEST TO SEE IF AN EMBEDDED
C    OR A TRAILING BLANK EXISTS.IF SO CONCATONATE. IF NOT THEN TEST TO
C    SEE IF THE QUANTITY STARTS WITH A SYMBOL.IF SO - ERROR. IF NOT
C    THEN DETERMINE IF ANY CHARACTER FOLLOWS THE QUANTITY TO TERMINATE
```

```
C     SEARCH AND CONCATENATION.
C
         IF( ( ICHS .GE. IC0 .AND. ICHS .LE. IC9 ) .OR.
     &       ( ICHS .GE. ICA .AND. ICHS .LE. ICZ ) ) THEN
C
           IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
             TEXT( 1:1 ) = STRING( I:I )
             IL = 1
           ELSE
             TEXT = TEXT( 1:IL )//STRING( I:I )
             IL = IL + 1
           ENDIF
C
         ELSE IF( STRING( I:I ) .EQ. ' ' .AND. TEXT( 1:1 ) .NE. ' ') THEN
C
             TEXT = TEXT( 1:IL )//STRING( I:I )
               IL = IL + 1
C
         ELSE IF( STRING( I:I ) .EQ. '(' .OR. STRING( I:I ) .EQ. ')' )
     &                                                              THEN
C
              IF( IPAREN .EQ. 0 ) THEN
C
                IF( STRING( I:I ) .EQ. '(' )THEN
                  TEXT = TEXT( 1:IL )//STRING( I:I )
                  IL = IL + 1
                  IPAREN = IPAREN + 1
                ELSE IF( STRING( I:I ) .EQ. ')' ) THEN
                  PRINT *,' ERROR - MISSING LEFT PARENTHESIS ',
     &                                        STRING(I-9:I)
                  PRINT *, ' '
                  ISTAT = 1
                  GO TO 9999
                ENDIF
C
              ELSE IF( IPAREN .EQ. 1 )THEN
C
                IF( STRING( I:I ) .EQ. '(' ) THEN
                  PRINT *,' ERROR - MISSING RIGHT PARENTHESIS ',
     &                                       STRING( I-9:I )
                  PRINT *,' '
                  ISTAT = 1
                  GO TO 9999
                ELSE IF( STRING( I:I ) .EQ. ')' )THEN
                  TEXT = TEXT( 1:IL )//STRING( I:I )
                  IL = IL + 1
                  IPAREN = IPAREN + 1
                ENDIF
C
              ELSE
C
                PRINT *,' ERROR - POSSIBLY TOO MANY PARENTHESIS ',
     &                                        STRING( I-9:I)
                PRINT *,' '
                ISTAT = 1
                GO TO 9999
C
              ENDIF
C
          ELSE IF( STRING( I:I ) .NE. ' ' .AND. TEXT( 1:1 ) .EQ. ' ') THEN
C
```

```
            IF( STRING( I:I ) .NE. ',' ) THEN
              PRINT *,' ERROR - QUANTITY STARTING WITH A SYMBOL ',
     &                                          STRING( I:I+7 )
              PRINT *,' '
              ISTAT = 1
              GO TO 9999
            ENDIF
C
          ELSE
              IF( STRING( I:I ) .NE. ' '.AND. TEXT(1:1) .NE. ' ') GO TO 30
          ENDIF
C
 20     CONTINUE
C
        IF( NDS .EQ. 0 ) THEN
         L = L + 1
         GO TO 9999
        ENDIF
C
 30     CONTINUE
C
C   SET POINTER TO LAST POSITION WHICH IS WHERE THE NEXT CHARACTER
C      BEGINS( MOST LIKELY AN EQUAL SIGN ).
C
        NL = I
        LT = ILEN( TEXT,NTEXT )
C
C   AFTER FINDING PARENTHESIS THIS INDICATES THAT AN ARRAY ELEMENT IS
C     BEING SPECIFIED. THEREFORE,CONVERT THE CHARACTER STRING WITHIN THE
C     PARENTHESIS TO AN INTEGER NUMBER.
C
        IF( IPAREN .EQ. 0 ) GO TO 35
C
        IF( IPAREN .NE. 2 ) THEN
          PRINT *,' ERROR - PARENTHESIS MISSING IN THE STRING ',TEXT(1:LT)
          PRINT *,' '
          ISTAT = 1
          GO TO 9999
        ELSE
          CALL ELEMNT( TEXT(1:LT) , LT , IPOINT , ISTAT )
          IF( ISTAT .NE. 0 ) GO TO 9999
          IPAREN = 0
        ENDIF
C
 35     CONTINUE
C
C   DETERMINE WETHER ANY BLANKS EMBEDDED WITHIN THE QUANTITY. IF SO
C     THEN ITS AN ERROR.
C
         N = INDEX( TEXT( 1:LT ) ,' ' )
C
        IF( N .NE. 0 ) THEN
          PRINT *,' ERROR - CAN NOT HAVE BLANKS EMBEDDED WITHIN THE NAME',
     &                                                TEXT( 1:LT )
          PRINT *,' '
          ISTAT = 1
          GO TO 9999
        ENDIF
C
C    SEARCH FOR THE QUANTITY IN THE LIST NAMLIS. IF NOT FOUND - ERROR
C
```

```
      DO 100 I = 1,NVAR
        IF( TEXT( 1:LT ) .EQ. NAMLIS( I ) ) GO TO 200
 100  CONTINUE
C
      PRINT *,' ERROR - COULD NOT FIND > ',TEXT(1:LT),' < IN NAMELIST '
      PRINT *,' '
      ISTAT = 1
      GO TO 9999
C
 200  CONTINUE
C
      INL = I
      IEQUAL = 0
      NUM = 0
      ITEXT = 0
C
        IF( IPOINT .LE. NDIM2( INL ) .AND. IPOINT .GE. NDIM1( INL ) )
     &                                                              THEN
          ITEXT = IPOINT - NDIM1( INL )
          IPOINT = 1
C
        ELSE
C
          PRINT *,' ERROR - SPECIFIED ARRAY ELEMENT OUT OF BOUNDS ',
     &            'FOR THE ARRAY ',NAMLIS( INL ) ,' .'
          PRINT *,'         THE LOWER AND UPPER BOUNDS ARE ',
     &            NDIM1( INL ) , NDIM2( INL )
          PRINT *,'         CHANGE THE ARRAY ELEMENT ',IPOINT,' TO FIT',
     &            ' WITHIN THE SPECIFIED LIMITS. '
          ISTAT = 1
          GO TO 9999
C
        ENDIF
C
C  FIND THE CHARACTER,INTEGER OR REAL STRING AND STORE IN THE
C    APPROPRIATE LOCATION TO BE LATER USED BY THE MAIN ROUTINE.
C
 210  CONTINUE
C
        NJ = ISIZ( NDIM1(INL) , NDIM2(INL) )
C
      NTCI = NCHAR + NINTG
C
      IF( INL .LE. NCHAR ) THEN
C
        POINTR = CPOS( INL )
        CALL FNDCHR( STRING,ICHR( POINTR ),ITEXT,LS,LT,NL,NJ,IEQUAL,
     &               NUM,ISTAT )
C
        IF( ISTAT .EQ. 1 ) GO TO 9999
C
      ELSE IF( INL .GT. NCHAR .AND. INL .LE. NTCI ) THEN
C
        II = INL - NCHAR
        POINTR = IPOS( II )
        CALL FNDINT( STRING , INTGR( POINTR ) , CDATA , IDAT , NDAT ,
     &               NIDAT , ITEXT , LS , LT , NL , NJ , IEQUAL ,
     &               NUM , ISTAT )
C
        IF( ISTAT .EQ. 1 ) GO TO 9999
C
```

```
      ELSE IF( INL .GT. NTCI ) THEN
C
        II = INL - NTCI
        POINTR = RPOS( II )
        CALL FNDRL( STRING , REALS( POINTR ) , CDATA , RDAT , NDAT ,
     &                NRDAT , ITEXT , LS , LT , NL , NJ , IEQUAL ,
     &                NUM , ISTAT )
C
        IF( ISTAT .EQ. 1 ) GO TO 9999
C
      ENDIF
C
C  IF THE POINTER IS AT THE END OF THE INPUT STRING i.e., THE PARSING
C   OF THE INPUT STRING HAS BEEN COMPLETED THEN GO TO 9999 TO THE MAIN
C   ROUTINE, OTHERWISE, GO TO 9999 TO THE BEGINNING AND SEARCH FOR THE
C   NEXT QUANTITY, ETC.
C
      DO 215 I = 1,NTEXT
 215    TEXT( I:I ) = ' '
C
      IF ( NL .LT. LS ) GO TO 10
C
C.....................................................................
C
C  IF THE SECOND DOLLAR SIGN WAS NOT FOUND THEN INCREMENT THE LINE
C    COUNTER('L'), OTHERWISE, RESET THE LINE COUNTER TO ONE(1) AND THE
C    ECHO PARAMETER TO ZERO(0).
C.....................................................................
C
      IF( NDS .EQ. 0 ) THEN
        L = L + 1
        RETURN
      ENDIF
C
 9999 CONTINUE
C
        L = 1
        IECHO = 0
        ITEXT = 0
C
      RETURN
      END
```

```
      SUBROUTINE FNDCHR( STRING , IC , ITEXT , LS , LT , L , NJ ,
     &                   IEQUAL , NUM , ISTAT )
C*********************************************************************
C
C  THIS SUBROUTINE FINDS A CHARACTER VALUE AND STORES IT IN THE
C     APPROPRIATE ARRAY( 'IC' ).
C
C  INPUT:
C
C        STRING - CHARACTER STRING OF LENGTH 'LS'.
C
C            LS - LENGTH OF CHARACTER STRING.
C
C            LT - LENGTH OF THE QUANTITY WHICH IS EQUIVALENCED TO THE
C                 ARRAY 'IC'.
C
C            IC - CHARACTER ARRAY CONTAINING THE INPUT CHARACTER VALUE.
C                 DIMENSIONED TO 'NJ'.
C
C             L - POINTER IN THE STRING 'STRING'. AT THE END OF THE
C                 ROUTINE IT'S UPDATED TO RETURN THE NEW POSITION.
C
C            NJ - DIMENSION OF THE ARRAY IC
C
C*********************************************************************
C
      CHARACTER*(*) STRING , IC
      CHARACTER*20 TEXT
C
      PARAMETER( NTEXT = 20 )
C
      DIMENSION IC( NJ )
C
C*********************************************************************
C*********************************************************************
C
      IC0 = ICHAR( '0' )
      IC9 = ICHAR( '9' )
C
      DO 5 I = 1,NTEXT
 5      TEXT( I:I ) = ' '
C
         IQ2 = 0
C
          IL = L - LT - 1
C
 10   CONTINUE
C
      DO 100 I = L,LS
C
C  CHECK FOR BLANKS. IF SO - INCREMENT.
C
        IF( STRING( I:I ) .EQ. ' ' ) GO TO 100
C
C LOOK FOR AN EQUAL SIGN AFTER THE QUANTITY. IF NOT THERE OR OTHER
C    CHARACTER IS THERE( EXCLUDING BLANKS ) THEN IT'S AN ERROR.
C
        IF( STRING( I:I ) .EQ. '=' ) THEN
          IEQUAL = 1
          GO TO 100
        ENDIF
```

```
C
C  IF EQUAL SIGN FOUND THEN NOW SEARCH AND FIND THE CHARACTER NEEDED FOR
C   THE INPUT.
C
      IF( IEQUAL .EQ. 1 ) THEN
C
C IF A COMMA FOLLOWS AN EQUAL SIGN THEN IT'S AN ERROR.
C
        IF( STRING( I:I ) .EQ. ',' ) THEN
          IF( ITEXT .EQ. 0 ) THEN
            PRINT *,' ERROR - COMMA IN WRONG POSITION ',
     3                        STRING( IL:I ),' <-'
            PRINT *,' '
            ISTAT = 1
            RETURN
          ENDIF
          GO TO 100
        ENDIF
C
        IF( STRING( I:I ) .EQ. '$' ) THEN
C
          IF( ITEXT .EQ. 0 ) THEN
            PRINT *,' ERROR - DOLLAR SIGN IN WRONG POSITION ',
     2                        STRING( IL:I ),'<-'
            PRINT *,' '
            ISTAT = 1
            RETURN
          ENDIF
C
          L = I
          RETURN
C
        ENDIF
C
C CHECK FOR A QUOTE - INDICATES CHARACTER VALUE AHEAD AND FIND THE
C      CHARACTER STRING USING ROUTINE 'CHRVAL'.
C
        ICHS = ICHAR( STRING( I:I ) )
C
        IF( STRING( I:I ) .EQ. '''' )THEN
C
          CALL CHRVAL( STRING , I , LS , IQ2 , ISTAT )
          IF( ISTAT .EQ. 1 ) RETURN
C
          ML = IQ2 - I - 1
C
C  NUM = 0 INDICATES THAT THE CHARACTER VALUE WILL NOT BE MULTIPLIED BY
C     AN INTEGER. THERFORE, CHARACTER VALUES WILL BE EXPECTED.
C
          IF( NUM .EQ. 0 ) THEN
            ITEXT = ITEXT + 1
C
C  ITEXT COUNTS THE NUMBER OF CHARCATER VALUES. IF ITS GREATER THAN THE
C     SPECIFIED DIMENSION OF THE ARRAY THEN IT'S AN ERROR.
C
            IF( ITEXT .GT. NJ ) THEN
              PRINT *,' ERROR - TOO MANY VALUES FOR ',STRING(IL:L )
              PRINT *,' '
              ISTAT = 1
              RETURN
            ENDIF
```

```
C
              IC( ITEXT ) = STRING( I+1:IQ2-1 )
C
            ELSE
C
              DO 20 J = ITXTO,ITEXT
                IC( J ) = STRING( I+1:IQ2-1 )
 20           CONTINUE
C
              NUM = 0
            ENDIF
C
C  SET NEW POINTER POSITION WITHIN 'STRING' AND GO BACK TO THE BEGINNING
C
            L = IQ2 + 1
            GO TO 10
C
C DETERMINE IF A CHARACTER STRING IS PRECEDED BY AN INTEGER AND A
C    MULTIPLICATION SIGN INDICATING THAT THE CHARACTER VALUE WILL BE
C    STORED IN THE ARRAY 'IC' THE AMOUNT GIVEN BY THE INTEGER.
C
          ELSE IF( ICHS .GE. IC0 .AND. ICHS .LE. IC9 ) THEN
C
              IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
                TEXT( 1:1 ) = STRING( I:I )
                JL = 1
              ELSE
                TEXT = TEXT( 1:JL )//STRING( I:I )
                JL = JL + 1
              ENDIF
C
          ELSE IF( STRING( I:I ) .EQ. '*' ) THEN
C
               IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
                 PRINT *,' ERROR - NEED AN INTEGER TO MULTIPLY ',
     3                   'CHARACTER VALUE ',STRING( IL:I+7 ),' <-'
                 PRINT *,' '
                 ISTAT = 1
                 RETURN
               ELSE
                 CALL CONVER( TEXT( 1:JL ) , VAR , ISTAT )
                 IF( ISTAT .EQ. 1 ) RETURN
C
                 ITXTO = ITEXT + 1
                 ITEXT = ITEXT + INT( VAR )
C
                 IF( ITEXT .GT. NJ ) THEN
                   PRINT *,' ERROR- TOO MANY VALUES FOR ',STRING(IL:I+7)
                   PRINT *,' '
                   ISTAT = 1
                   RETURN
                 ENDIF
C
                 NUM = 1
C
                 DO 30 J = 1,NTEXT
 30                TEXT( J:J ) = ' '
C
               ENDIF
C
         ELSE IF( TEXT(1:1) .NE. ' ' .AND. STRING( I:I ) .NE. '*' ) THEN
```

```
C
                PRINT *,' WARNING - POSSIBLE ERROR ',STRING( IL:I ),'<-'
                PRINT *,' '
C
              L = I
              RETURN
C
            ELSE
C
              IF( ITEXT .LE. NJ ) THEN
                L = I
                RETURN
              ENDIF
C
            ENDIF
C
          ELSE
C
            PRINT *,' ERROR - WRONG CHARACTER OR EQUAL SIGN MISSING ',
     &              STRING( IL:I+1 )
            PRINT *,' '
            ISTAT = 1
            RETURN
C
          ENDIF
C
  100   CONTINUE
C
        L = LS
C
        RETURN
        END
```

```
      SUBROUTINE FNDINT( STRING , IC , CDATA , IDAT , NDAT , NIDAT ,
     &                   ITEXT , LS , LT , L , NJ , IEQUAL , NUM ,
     &                   ISTAT )
C*****************************************************************************
C
C  THIS SUBROUTINE FINDS AN INTEGER VALUE FROM THE INPUT CHARACTER
C     STRING 'STRING'. THIS ROUTINE ALLOWS THE USER TO SPECIFY
C     ALPHANUMERIC CHARACTERS FOR THE QUANTITY WHICH WILL THEN BE
C     CONVERTED TO UNIQUE INTEGER VALUES.THIS PART IS ACCOMPLISHED IN
C     SUBROUTINE CCI.
C
C    INPUT/OUTPUT:
C
C         STRING - CHARACTER VARIABLE OF LENTH 'LS' CONTAINING THE
C                  INPUT CHARACTER STRING.
C
C             LS - LENGTH OF CHARACTER STRING.
C
C          LTEXT - LENTGHT OF THE QUANTITY WHICH IS EQUIVALEND TO THE
C                  ARRAY 'IC'.
C
C             IC - INTEGER ARRAY CONTAINING THE APPROPRIATE VALUES FOR
C                  THE SPECIFIED QUANTITY.
C
C              L - POINTER IN THE STRING 'STRING', AT THE END OF THE
C                  ROUTINE 'L' IS UPDATED TO RETURN THE NEW POSITION.
C
C             NJ - DIMENSION OF THE ARRAY 'IC'.
C
C          CDATA - CHARACTER ARRAY CONTAINING THE CHARACTER
C                  REPRESENTATION OF VARIOUS VALUES.
C                  CDATA DIMENSIONED TO 'NDAT'.
C
C           IDAT - INTEGER ARRAY CONTAINING THE RESPECTIVE INTEGER
C                  VALUES. DIMENSIONED TO 'NIDAT'.
C
C           NDAT - TOTAL NUMBER OF VALUES( INTEGER AND REAL ) IN THE
C                  ARRAY 'CDATA'.
C
C          NIDAT - TOTAL NUMBER OF INTEGER VALUES IN THE ARRAY 'IDAT'.
C
C*****************************************************************************
C
      CHARACTER*(*) STRING , CDATA
      CHARACTER*20 TEXT
C
      PARAMETER( NTEXT = 20 )
C
      DIMENSION IC( NJ ) , CDATA( NDAT ) , IDAT( NIDAT )
C
C*****************************************************************************
C*****************************************************************************
C
      DO 5 I = 1,NTEXT
    5   TEXT( I:I ) = ' '
C
      IC0 = ICHAR( '0' )
      IC9 = ICHAR( '9' )
      ICMS = ICHAR( '-' )
      ICPS = ICHAR( '+' )
C
```

```
          IQ2 = 0
C
           IL = L - LT -1
C
 10    CONTINUE
C
       DO 100 I = L,L3
C
         IF( STRING( I:I ) .EQ. ' ' ) GO TO 100
C
         IF( STRING( I:I ) .EQ. '=' ) THEN
           IEQUAL = 1
           GO TO 100
         ENDIF
C
C   IF THE EQUAL SIGN HAS BEEN FOUND THEN FIND THE INTEGER VALUES.
C
         IF( IEQUAL .EQ. 1 ) THEN
C
C   DETERMINE IF THE CHARACTER IS A NUMBER. IF SO, CONCATONATE.
C
           ICHS = ICHAR( STRING( I:I ) )
C
           IF( ( ICHS .GE. IC0 .AND. ICHS .LE. IC9 ) .OR.
     &           ICHS .EQ. ICPS .OR. ICHS .EQ. ICMS ) THEN
C
             IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
               TEXT( 1:1 ) = STRING( I:I )
               JL = 1
             ELSE
               TEXT = TEXT( 1:JL )//STRING( I:I )
               JL = JL + 1
             ENDIF
C
C   CHECK FOR COMMA. IF NOTHING IN TEXT - ERROR.OTHERWISE,CONVERT STRING
C     TO INTEGER.
C
           ELSF IF( STRING( I:I ) .EQ. ',' .OR. STRING( I:I ) .EQ. '$' )
     &                                                              THEN
C
               IF( IQ2 .EQ. 9999 ) THEN
C
                 IQ2 = 0
C
                 IF( STRING( I:I ) .EQ. ',' ) GO TO 100
                 IF( STRING( I:I ) .EQ. '$' ) THEN
                   L = I
                   RETURN
                 ENDIF
C
               ENDIF
C
               IF( TEXT( 1:1 ) .EQ. ' ' .AND. STRING( I:I ) .EQ. ',' )
     &                                                              THEN
                 PRINT *,' ERROR - NO VALUE FOUND PRIOR TO REACHING A ',
     &                   'COMMA ',STRING( IL:I ),' <-'
                 PRINT *,' '
                 ISTAT = 1
                 RETURN
               ENDIF
C
```

```
            IF( TEXT( 1:1 ) .EQ. ' ' .AND. STRING( I:I ) .EQ. '$' )
     &                                                          THEN
              PRINT *,' ERROR - NO VALUE FOUND PRIOR TO REACHING A ',
     &                'DOLLAR SIGN ',STRING( IL:I ),' <-'
              PRINT *,' '
              ISTAT = 1
              RETURN
            ENDIF
C
C   NUM = 0 INDICATES NO MULTIPLICATION OF VALUE BY INTEGER.
C
            IF( NUM .EQ. 0 ) THEN
C
              ITEXT = ITEXT + 1
C
              IF( ITEXT .GT. NJ ) THEN
                PRINT *,' ERROR - TOO MANY VALUES FOR ',STRING( IL:I )
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
              CALL CONVER( TEXT( 1:JL ) , VAR , ISTAT )
C
              IF( ISTAT .EQ. 1 ) RETURN
C
              IC( ITEXT ) = INT( VAR )
C
              DO 20 J = 1,NTEXT
 20             TEXT( J:J ) = ' '
C
            ELSE
C
              CALL CONVER( TEXT( 1:JL ) , VAR , ISTAT )
C
              IF( ISTAT .EQ. 1 ) RETURN
C
              IVAR = INT( VAR )
C
              DO 30 J = ITXTO,ITEXT
                IC( J ) = IVAR
 30             CONTINUE
C
              NUM = 0
C
              DO 40 J = 1,NTEXT
 40             TEXT( J:J ) = ' '
C
            ENDIF
C
C   RESET POINTER AND GO TO BEGINNING TO FIND ANOTHER VALUE( IF ANY )
C
            IF( STRING ( I:I ) .EQ. '$' ) THEN
               L = I
               RETURN
            ENDIF
C
            L = I + 1
            GO TO 10
C
C IF '*' FOUND THEN CONVERT TEXT TO INTEGER AND MULTIPLY BY THE UPCOMING
```

```
C       VALUE.
C
          ELSE IF( STRING( I:I ) .EQ. '*' ) THEN
               IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
                 PRINT *,' ERROR - INTEGER REQUIRED ' , STRING( IL:I ),
     &                                                          ' <-'
                 PRINT *,' '
                 ISTAT = 1
                 RETURN
               ENDIF
C
               KL = ILEN( TEXT , NTEXT )
C
               N = INDEX( TEXT( 1:KL ) , '-' )
C
               IF( N .NE. 0 ) THEN
                 PRINT *,' ERROR - CAN NOT MULTIPLY BY NEGATIVE INTEGER ',
     &                    TEXT( 1:KL )
                 PRINT *,' '
                 ISTAT = 1
                 RETURN
               ENDIF
C
               CALL CONVER( TEXT( 1:KL ) , VAR , ISTAT )
C
               IF( ISTAT .EQ. 1 ) RETURN
C
               ITXTO = ITEXT + 1
               ITEXT = ITEXT + INT( VAR )
C
               IF( ITEXT .GT. NJ ) THEN
                 PRINT *,' ERROR - TOO MANY VALUES FOR ',STRING( IL:I )
                 PRINT *,' '
                 ISTAT = 1
                 RETURN
               ENDIF
C
                NUM = 1
                DO 60 J = 1,NTEXT
 60               TEXT( J:J ) = ' '
C
C  IF QUOTE FOUND THEN FIND THE CHARACTER VALUE CONVERT IT TO A UNIQUE
C    NUMBER USING ROUTINE 'CCI'.
C
          ELSE IF( STRING( I:I ) .EQ. '''' ) THEN
C
               CALL CHRVAL( STRING , I , L5 , I02 , ISTAT )
C
               IF( ISTAT .EQ. 1 ) RETURN
C
               CALL CCI ( STRING( I:I02 ) , IVAR , CDATA ,IDAT , NDAT ,
     &                                                 NIDAT , ISTAT )
C
                IF( ISTAT .EQ. 1 ) RETURN
C
               IF( NUM .EQ. 0 ) THEN
C
                 ITEXT = ITEXT + 1
C
                 IF( ITEXT .GT. NJ ) THEN
                   PRINT *,' ERROR - TOO MANY VALUES FOR ',STRING( IL:I )
```

```
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
              IC( ITEXT ) = IVAR
C
            ELSE
C
              DO 80 J = ITXTO,ITEXT
                IC( J ) = IVAR
 80           CONTINUE
C
              NUM = 0
C
            ENDIF
C
            L = IQ2 + 1
            IQ2 = 9999
C
            GO TO 10
C
        ELSE IF( TEXT(I:I) .NE. ' ' .AND. STRING( I:I ) .NE. '*' ) THEN
C
              PRINT *,' WARNING - POSSIBLE ERROR ',STRING( IL:I ),'<-'
              PRINT *,' '
C
              L = I
              RETURN
C
        ELSE
C
             IF( ITEXT .LE. NJ ) THEN
               L = I
               RETURN
             ENDIF
C        ENDIF
C
       ELSE
C
          PRINT *,' ERROR - WRONG CHARACTER OR EQUAL SIGN MISSING ',
     &                                              STRING( IL:I ),' <-'
          PRINT *,' '
          ISTAT = 1
          RETURN
C
       ENDIF
 100   CONTINUE
C
       L = LS
C
       RETURN
       END
```

```
      SUBROUTINE FNDRL( STRING , IC , CDATA , RDAT , NDAT , NRDAT ,
     &                  ITEXT , LS , LT , L , NJ , IEQUAL , NUM ,
     &                  ISTAT )
C*********************************************************************
C
C  THIS SUBROUTINE FINDS AN INTEGER VALUE FROM THE INPUT CHARACTER
C     STRING 'STRING'. THIS ROUTINE ALLOWS THE USER TO SPECIFY
C     ALPHANUMERIC CHARACTERS FOR THE QUANTITY WHICH WILL THEN BE
C     CONVERTED TO UNIQUE INTEGER VALUES.THIS PART IS ACCOMPLISHED IN
C     SUBROUTINE CCI.
C
C    INPUT/OUTPUT:
C
C         STRING - CHARACTER VARIABLE OF LENTH 'LS' CONTAINING THE
C                  INPUT CHARACTER STRING.
C
C             LS - LENGTH OF CHARACTER STRING.
C
C          LTEXT - LENTGHT OF THE QUANTITY WHICH IS EQUIVALEND TO THE
C                  ARRAY 'IC'.
C
C             IC - INTEGER ARRAY CONTAINING THE APPROPRIATE VALUES FOR
C                  THE SPECIFIED QUANTITY.
C
C              L - POINTER IN THE STRING 'STRING'. AT THE END OF THE
C                  ROUTINE 'L' IS UPDATED TO RETURN THE NEW POSITION.
C
C             NJ - DIMENSION OF THE ARRAY 'IC'.
C
C          CDATA - CHARACTER ARRAY CONTAINING THE CHARACTER
C                  REPRESENTATION OF VARIOUS VALUES.
C                  CDATA DIMENSIONED TO 'NDAT'.
C
C           RDAT - REAL ARRAY CONTAINING THE RESPECTIVE REAL
C                  VALUES. DIMENSIONED TO 'NRDAT'.
C
C           NDAT - TOTAL NUMBER OF VALUES( INTEGER AND REAL ) IN THE
C                  ARRAY 'CDATA'.
C
C          NRDAT - TOTAL NUMBER OF REAL VALUES IN THE ARRAY 'RDAT'.
C                  .
C*********************************************************************
C
      CHARACTER*(*) STRING , CDATA
      CHARACTER*20 TEXT
C
      REAL IC
C
      PARAMETER( NTEXT = 20 )
C
      DIMENSION IC( NJ ) , CDATA( NDAT ) , RDAT( NRDAT )
C
C*********************************************************************
C*********************************************************************
C
      DO 5 I = 1,NTEXT
 5      TEXT( I:I ) = ' '
C
      IC0 = ICHAR( '0' )
      IC9 = ICHAR( '9' )
      ICMS = ICHAR( '-' )
```

```
      ICPS = ICHAR( '+' )
      ICPER = ICHAR( '.' )
C
         IQ2 = 0
C
          IL = L - LT - 1
C
 10   CONTINUE
C
      DO 100 I = L,LS
C
        IF( STRING( I:I ) .EQ. ' ' ) GO TO 100
C
        IF( STRING( I:I ) .EQ. '=' ) THEN
          IEQUAL = 1
          GO TO 100
        ENDIF
C
C  IF THE EQUAL SIGN HAS BEEN FOUND THEN FIND THE REAL VALUES.
C
        IF( IEQUAL .EQ. 1 ) THEN
C
C  DETERMINE IF THE CHARACTER IS A NUMBER. IF SO, DETERMINE WHETHER IT'S
C    A ZERO. IF SO, ERROR;OTHERWISE CONCATONATE.
C
          ICHS = ICHAR( STRING( I:I ) )
C
          IF( ( ICHS .GE. IC0 .AND. ICHS .LE. IC9 ) .OR.
     &          ICHS .EQ. ICMS .OR. ICHS .EQ. ICPS .OR.
     &                              ICHS .EQ. ICPER ) THEN
C
            IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
              TEXT( 1:1 ) = STRING( I:I )
              JL = 1
            ELSE
              TEXT = TEXT( 1:JL )//STRING( I:I )
              JL = JL + 1
            ENDIF
C
C  CHECK FOR COMMA. IF NOTHING IN TEXT - ERROR.OTHERWISE,CONVERT STRING
C    TO REAL.
C
          ELSE IF( STRING( I:I ) .EQ. ',' .OR. STRING( I:I ) .EQ. '$' )
     &                                                          THEN
C
            IF( IQ2 .EQ. 9999 ) THEN
C
              IQ2 = 0
C
              IF( STRING( I:I ) .EQ. ',' ) GO TO 100
              IF( STRING( I:I ) .EQ. '$' ) THEN
                L = I
                RETURN
              ENDIF
C
            ENDIF
C
            IF( TEXT( 1:1 ) .EQ. ' ' .AND. STRING( I:I ) .EQ. ',' )
     %                                                          THEN
              PRINT *,' ERROR - NO VALUE FOUND PRIOR TO REACHING A ',
     &                 'COMMA ',STRING( IL:I ),' <-'
```

```
                ISTAT = 1
                RETURN
              ENDIF
C
              IF( TEXT( 1:1 ) .EQ. ' ' .AND. STRING( I:I ) .EQ. '$' )
     &                                                          THEN
                PRINT *,' ERROR - NO VALUE FOUND PRIOR TO REACHING A ',
     &                  'DOLLAR SIGN ',STRING( IL:I ),' <-'
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
C  NUM = 0 INDICATES NO MULTIPLICATION OF VALUE BY INTEGER.
C
              IF( NUM .EQ. 0 ) THEN
C
                ITEXT = ITEXT + 1
C
                IF( ITEXT .GT. NJ ) THEN
                  PRINT *,' ERROR - TOO MANY VALUES FOR ',STRING( IL:I )
                  PRINT *,' '
                  ISTAT = 1
                  RETURN
                ENDIF
C
                CALL CONVER( TEXT( 1:JL ) , VAR , ISTAT )
C
                IF( ISTAT .EQ. 1 ) RETURN

                IC( ITEXT ) =  VAR
'
                DO 20 J = 1,NTEXT
 20               TEXT( J:J ) = ' '
C
              ELSE
C
                CALL CONVER( TEXT( 1:JL ) , VAR , ISTAT )
C
                IF( ISTAT .EQ. 1 ) RETURN
C
                DO 30 J = ITXTO,ITEXT
                  IC( J ) = VAR
 30               CONTINUE
C
                NUM = 0
C
                DO 40 J = 1,NTEXT
 40               TEXT( J:J ) = ' '
C
              ENDIF
C
C  RESET POINTER AND GO TO BEGINNING TO FIND ANOTHER VALUE( IF ANY )
C
              IF( STRING ( I:I ) .EQ. '$' ) THEN
                 L = I
                 RETURN
              ENDIF
C
              L = I + 1
              GO TO 10
```

```
C
C IF '*' FOUND THEN CONVERT TEXT TO INTEGER AND MULTIPLY BY THE UPCOMING
C       VALUE.
C
          ELSE IF( STRING( I:I ) .EQ. '*' ) THEN
              IF( TEXT( 1:1 ) .EQ. ' ' ) THEN
                PRINT *,' ERROR - INTEGER REQUIRED ' , STRING( IL:I ),
     &                                                          ' <-'
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
              KL = ILEN( TEXT , NTEXT )
C
              N = INDEX( TEXT( 1:KL ) , '-' )
C
              IF( N .NE. 0 ) THEN
                PRINT *,' ERROR - CAN NOT MULTIPLY BY NEGATIVE INTEGER ',
     &                  TEXT( 1:KL )
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
              N = INDEX( TEXT( 1:KL ) , '.' )
C
              IF( N .NE. 0 ) THEN
                PRINT *,' ERROR - LOOKING FOR AN INTEGER NOT A REAL ',
     &                  'NUMBER ', TEXT( 1:KL )
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
              CALL CONVER( TEXT( 1:KL ) , VAR , ISTAT )
C
              IF( ISTAT .EQ. 1 ) RETURN
C
              ITXTO = ITEXT + 1
              ITEXT = ITEXT + INT( VAR )
C
              IF( ITEXT .GT. NJ ) THEN
                PRINT *,' ERROR - TOO MANY VALUES FOR ',STRING( IL:I )
                PRINT *,' '
                ISTAT = 1
                RETURN
              ENDIF
C
               NUM = 1
C
               DO 60 J = 1,NTEXT
 60              TEXT( J:J ) = ' '
C
C   IF QUOTE FOUND THEN FIND THE CHARACTER VALUE CONVERT IT TO A UNIQUE
C     NUMBER USING ROUTINE 'CCR'.
C
          ELSE IF( STRING( I:I ) .EQ. '''' ) THEN
C
              CALL CHRVAL' STRING , I , LS , IQ2 , ISTAT )
C
```

```
            IF( ISTAT .EQ. 1 ) RETURN
C
          CALL CCR ( STRING( I:IQ2 ) : VAR , CDATA ,RDAT , NDAT ,
     &                                                 NRDAT , ISTAT )
C
            IF( ISTAT .EQ. 1 ) RETURN
C
          IF( NUM .EQ. 0 ) THEN
C
            ITEXT = ITEXT + 1
C
            IF( ITEXT .GT. NJ ) THEN
              PRINT *,' ERROR - TOO MANY VALUES FOR ';STRING( IL:I )
              PRINT *,' '
              ISTAT = 1
              RETURN
            ENDIF
C
            IC( ITEXT ) = VAR
C
          ELSE
C
            DO 80 J = ITXTO,ITEXT
              IC( J ) = VAR
 80         CONTINUE
C
            NUM = 0
C
          ENDIF
C
          L = IQ2 + 1
          IQ2 = 9999
C
          GO TO 10
C
        ELSE IF( TEXT(1:1) .NE. ' ' .AND. STRING( I:I ) .NE. '*' ) THEN
C
            PRINT *,' WARNING - POSSIBLE ERROR ';STRING( IL:I );'<-'
            PRINT *,' '
C
          L = I
          RETURN
C
        ELSE
C
           IF( ITEXT .LE. NJ ) THEN
             L = I
             RETURN
           ENDIF
C
        ENDIF
C
      ELSE
C
        PRINT *,' ERROR - WRONG CHARACTER OR EQUAL SIGN MISSING ',
     &                                          STRING( IL:I );' <-'
        PRINT *,' '
        ISTAT = 1
        RETURN
C
      ENDIF
```

```
 100  CONTINUE
C
      L = LS
C
      RETURN
      END
```

```
      SUBROUTINE CHRVAL( STRING , I , LS , IQ2 , ISTAT )
      CHARACTER*(*) STRING
C*********************************************************************
C
C  THIS SUBROUTINE FINDS A CHARACTER VALUE CHECKS IT AND MAKES SURE
C     THAT IT IS CORRECT. IT RETURNS THE VALUE OF THE SECOND QUOTE.
C
C*********************************************************************
C
            NC = LS - 1
            IQ2 = INDEX( STRING( I+1:NC ) , '''' )
C
            IF( IQ2 .EQ. 0 ) THEN
              PRINT *,'ERROR - MISSING SECOND QUOTE ', STRING( I:NC ),
     &                                                 ' <-'
              ISTAT = 1
              RETURN
            ENDIF
C
C CHECK FOR EMBEDDED BLANKS WITHIN THE TWO QUOTES.
C
            IQ2 = IQ2 + I
C
            N = INDEX( STRING( I:IQ2 ) , ' ' )
C
            IF( N .NE. 0 ) THEN
              PRINT *,' ERROR - CANNOT HAVE EMBEDDED BLANKS ',
     &                'IN CHARACTER VALUES ',
     &                                  STRING( I:IQ2 ) ,' <-'
              ISTAT = 1
              RETURN
            ENDIF
C
      RETURN
      END
```

```
      SUBROUTINE CONVER( TEXT , VAR , ISTAT )
C*********************************************************************
C
C  THIS SUBROUTINE WILL CONVERT AN ANSI REPRESENTATION OF A NUMBER TO
C     AN ACTUAL NUMBER( INTEGER OR REAL - EXCLUDING 'E' FORMAT ).
C
C   INPUT/OUTPUT:
C
C         TEXT - CHARACTER STRING WITH THE ANSI REPRESENTATION OF A
C                NUMBER
C
C          VAR - ACTUAL NUMERICAL VALUE BEING RETURNED. IT CAN BE EITHER
C                INTEGER OR REAL. FOR THE MOMENT A REAL NUMBER
C                REPRESENTED BY 'E' FORMAT IS NOT AVAILABLE.
C
C >>>>>NOTE: - THE ALGORITHM FOR THIS  ROUTINE PROVIDED BY
C                FRANK MANSFIELD OF NSWC,CA.
C
C*********************************************************************
C
      CHARACTER*(*) TEXT
      INTEGER DIGIT
C
      VAR = 0.0
C
      ICO = ICHAR( '0' )
C
       N = LEN( TEXT )
      NT = ILEN( TEXT ,`N )
C
      DO 10 I =1,NT
C
        IF( TEXT( I:I ) .EQ. ' ' ) GO TO 10
C
        IF( TEXT( I:I ) .EQ. '-' ) THEN
C
          NSTR = I + 1
          SIG = -1.0
C
        ELSE IF( TEXT( I:I ) .EQ. '+' ) THEN
C
          NSTR = I + 1
          SIG = 1.0
C
        ELSE
C
          NSTR = I
          SIG = 1.0
C
        ENDIF
C
        GO TO 20
C
 10   CONTINUE
C
      PRINT *,' ERROR - NO CHARACTER STRING FOUND IN TEXT =',TEXT(1:NT )
      ISTAT = 1
      RETURN
C
 20   CONTINUE
C
```

```
      NDEC = INDEX( TEXT( NSTR:NT ) , '.' )
C
      IF( NDEC .EQ. 0 ) THEN
        NDEC = NT+1
      ELSE
        NDEC = NDEC + NSTR - 1
      ENDIF
C
      TEN = 0.1
C
      DO 100 I = NDEC-1,NSTR,-1
C
        DIGIT = ICHAR( TEXT( I:I ) ) - IC0
          TEN = 10.*TEN
          VAR = VAR + TEN*DIGIT
C
 100  CONTINUE
C
      TEN = 1.0
C
      DO 200 I = NDEC+1,NT
C
        DIGIT = ICHAR( TEXT( I:I ) ) - IC0
          TEN = 0.1*TEN
          VAR = VAR + TEN*DIGIT
C
 200  CONTINUE
C
      VAR = VAR*SIG
C
      RETURN
      END
```

```
      SUBROUTINE CCI( TEXT , IVAR , CDATA , IDAT , NDAT , NIDAT , ISTAT)
C*********************************************************************
C
C    THIS SUBROUTINE CONVERTS CHARATER INFORMATION INTO UNIQUE INTEGER
C       VALUES BASE UPON THE ALLOWABLE INPUTS FOR THE DESIRED QUANTITY.
C
C*********************************************************************
C
      CHARACTER*(*) TEXT , CDATA
C
      DIMENSION CDATA( NDAT ) , IDAT( NIDAT )
C
      N = LEN( TEXT )
      NT = ILEN( TEXT , N )
C
      DO 10 I = 1,NDAT
        IF( TEXT( 1+1:NT-1 ) .EQ. CDATA( I ) ) GO TO 20
 10   CONTINUE
C
      PRINT *,' ERROR - ',TEXT( 1:NT ),' NOT FOUND IN CDATA ARRAY '
      ISTAT = 1
      RETURN
C
 20   CONTINUE
C
      IF( I .GT. NIDAT ) THEN
        PRINT *,' ERROR - INCORRECT CHARACTER STRING , EXPECTING ',
     &          'ONE OF THE,E '
        PRINT *,( CDATA( J ) ,J =1,NIDAT )
        PRINT *,' INSTEAD RECEIVED THIS ',TEXT( 1:NT )
        ISTAT = 1
        RETURN
      ENDIF
C
      IVAR = IDAT( I )
C
      RETURN
      END
```

```
      SUBROUTINE CCR( TEXT , VAR , CDATA , RDAT , NDAT , NRDAT , ISTAT )
C*********************************************************************
C
C    THIS SUBROUTINE CONVERTS CHARATER INFORMATION INTO UNIQUE REAL
C      VALUES BASED UPON THE ALLOWABLE INPUTS FOR THE DESIRED QUANTITY.
C
C*********************************************************************
C
      CHARACTER*(*) TEXT , CDATA
C
      DIMENSION CDATA( NDAT ) , RDAT( NRDAT )
C
      N = LEN( TEXT )
      NT = ILEN( TEXT , N )
C
      DO 10 I = 1,NDAT
        IF( TEXT( 1+1:NT-1 ) .EQ. CDATA( I ) ) GO TO 20
 10   CONTINUE
C
      PRINT *,' ERROR - ',TEXT( 1:NT ),' NOT FOUND IN CDATA ARRAY '
      ISTAT = 1
      RETURN
C
 20   CONTINUE
C
      NIDAT = NDAT - NRDAT
      IF( I .LE. NIDAT ) THEN
C
        N = NIDAT + 1
        IF( NIDAT .EQ. NDAT ) N = NDAT
C
        PRINT *,' ERROR - INCORRECT CHARACTER VALUE , EXPECTING ' ,
     &          'ONE OF THESE '
        PRINT *,( CDATA( J ) ,J =N,NDAT )
        PRINT *,' INSTEAD RECEIVED THIS ',TEXT( 1:NT )
        ISTAT = 1
        RETURN
      ENDIF
C
      II = I - NIDAT
      VAR = RDAT( II )
C
      RETURN
      END
```

```
      SUBROUTINE ELEMNT( TEXT , LT , IVAR , ISTAT )
C*******************************************************************
C
C  THIS ROUTINE CONVERTS A CHARACTER STRING WITHIN A SET OF PARENTHESIS
C    TO AN INTEGER NUMBER INDICATING THE POSITION WITHIN THE SPECIFIED
C    ARRAY.
C
C    INPUT/OUTPUT:
C
C             TEXT - CHARTACTER STRING CONTAINING THE ARRAY NAME AND
C                     THE ARRAY ELEMENT.
C
C               LT - CHARACTER LENGTH OF THE VARIABLE.
C
C             IVAR - INTEGER NUMBER ( POSITIVE OR NEGATIVE ) BEING
C                    RETURNED TO THE CALLING ROUTINE. INDICATING THE
C                    ARRAY ELEMENT DESIRED.
C
C            ISTAT - ERROR FLAG ; 0 - NO ERROR; 1 - ERROR
C
C*******************************************************************
C
      CHARACTER*(*) TEXT
      CHARACTER*32 NUM
C
      DO 1 I = 1,32
 1      NUM( I:I ) = ' '
C
      ICO = ICHAR( '0' )
      IC9 = ICHAR( '9' )
C
      ICMIN = ICHAR( '-' )
      ICPLS = ICHAR( '+' )
C
C  LOCATE POSITION OF THE SET OF PARENTHESIS. THEN, USE THESE VALUES
C   IN THE DO LOOP TO FIND THE CHARACTER STRING 'NUM' WHICH WILL THEN
C   BE CONVERTED TO AN INTEGER VALUE 'IVAR' )
C
      NL = INDEX( TEXT , '(' )
      NR = INDEX( TEXT , ')' )
C
      LT = NL - 1
C
      DO 10 I = NL+1 , NR-1
C
        IF( TEXT( I:I ) .EQ. ' ' ) GO TO 10
C
        ICHS = ICHAR( TEXT( I:I ) )
C
        IF( ( ICHS .GE. ICO .AND. ICHS .LE. IC9 )  OR.
     &        ICHS .EQ. ICMIN .OR. ICHS .EQ. ICPLS ) THEN
C
           IF( NUM( 1:1 ) .EQ. ' ' ) THEN
             NUM( 1:1 ) = TEXT( I:I )
             IN = 1
           ELSE
             NUM = NUM( 1:IN )//TEXT( I:I )
             IN = IN + 1
           ENDIF
C
        ELSE
```

```
C
            PRINT *,' ERROR - INCORRECT CHARACTER WITHIN PARENTHESIS.',
     &              ' CHECK ', TEXT( 1:I ),' <='
            PRINT *,' '
            ISTAT = 1
            RETURN
C
          ENDIF
C
 10     CONTINUE
C
        CALL CONVER( NUM , VAR , ISTAT )
C
        IF( ISTAT .NE. 0 ) RETURN
C
        IVAR = INT( VAR )
C
        RETURN
        END
```

```
      FUNCTION ILEN( TEXT , N )
      CHARACTER*(*) TEXT
C
      DO 100 I = N,1,-1
        IF( TEXT( I:I ) .NE. ' ' ) GO TO 200
 100  CONTINUE
C
      ILEN = 0
      PRINT *,' WARNING - POSSIBLE ERROR. LENGTH OF TEXT = ',TEXT,
     &        ' IS ZERO .'
      RETURN
C
 200  CONTINUE
C
      ILEN = I
C
      RETURN
      END


      FUNCTION ISIZ( N1 , N2 )
C
      ISIZ = 0
C
      DO 10 I = N1 , N2
        ISIZ = ISIZ + 1
 10   CONTINUE
C
      RETURN
      END
```